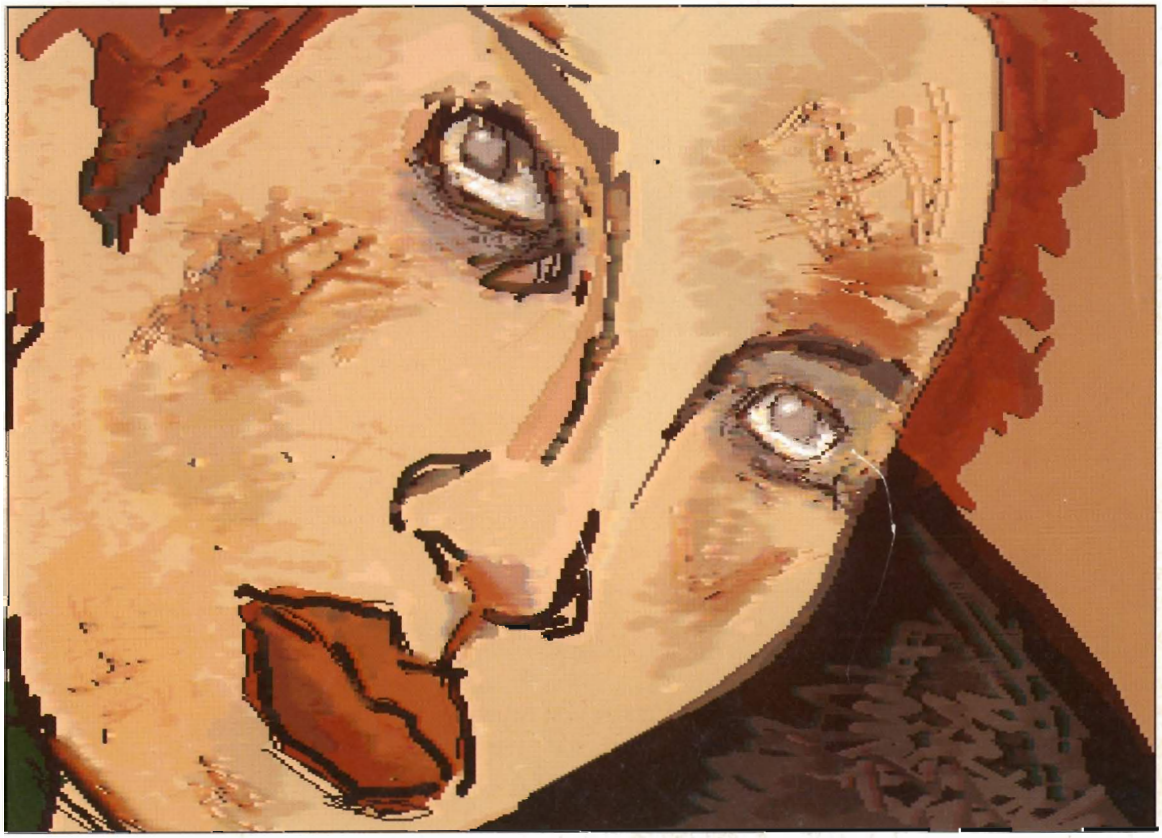
MAY 1989: VOLUME 2, ISSUE 2

THE MAGAZINE FOR AMIGA PROGRAMMERS

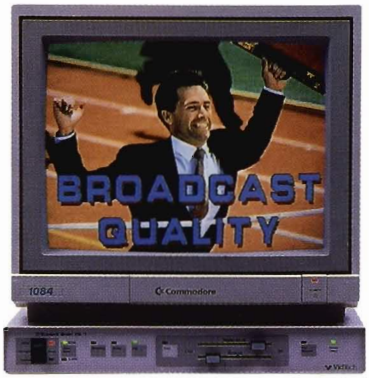
Canada $4.25
USA $3.50

FOR THE Amiga Transactor

- Getting a handle on file systems *by John Toebes*
- Language grammars using *Bison/Yacc*, part 2 *by Eric Giguere*
- MIDI programming primer *by Daniel Brookshier*
- Mastering the Raster: video presentations *by Derek Grime*
- Assembly language programming *by Jim Butterfield*
- C coding styles *by Ben Blish*
- An introduction to four DOS shells *by Mike Hubbartt*
- Amiga Dispatches: a board-swapping nightmare! *by Don Curtis*
- Speech routines in Modula-2 *by Anthony Bryant*
- **Reviews:** Interfont, DevPac, Hisoft BASIC Professional, and three hard drive backup utilities



Portrait - Egon by Christine Delezenne



Coming To A Screen Near You.

If you're ready to become your own studio producer, there's only one desktop video system that will bring NTSC and S-VHS to your Amiga's screen. The Scanlock system by VidTech. Other systems give you a genlock that's "either-or." They either support NTSC or S-VHS. But not both.

Two For One

With Scanlock, you get two genlocks for the price of one. So you can migrate from NTSC to the newer S-VHS format without investing in another system. We also offer a Scanlock model that is compatible with the International PAL TV format, including its new S-VHS version.

Scanlock's dual-genlock system gives you broadcast-quality video at consumer prices. And our system offers all the bells and whistles, including front-panel fingertip controls, multiple monitor connections for viewing and editing, and independent fade controls for both reference and Amiga videos. If your Amiga is supporting a load of peripherals, we provide a connection that lets you use an optional external power supply.

Scanlock is a breeze to use. A "glitch-free" vertical interval switch lets you insert, reverse and remove reference video and Amiga graphics at the touch of a finger. And you can switch between NTSC, S-VHS and analog RGB formats without moving cable connectors. When you're done editing, you can resume normal Amiga operation with the flip of a switch.

Don't Get Locked Out

So if you're ready to become your own producer, choose the desktop video system that doesn't lock you into one genlock — the Scanlock system by VidTech. Coming to a screen near you.

Call or write us today: VidTech International, Inc., 2822 NW 79th Ave., Miami, Florida 33122. Telephone 800-727-2261 or 305-477-2228. Fax 305-591-1651. Prices:NTSC-$995, PAL-$1,095. Distributed by Southern Technologies, Dallas, Texas.

VidTech

The Inovatronics Assembly Line:

InovaTools 2

Advanced Assembler Functions

C.A.P.E.68k
Version 2

A Technical Knockout!

Inovatronics understands assembly language. All of our in-house Amiga software development is done in assembler. We know what programmers desire and *require* from an assembler and C.A.P.E.68k delivers: integration, optimization and SPEED!

We also understand that getting started in assembler is difficult for programmers crossing over from a high level language such as "C", or Modula-2, especially on the Amiga! Enter InovaTools 2, a package of expertly programmed routines designed to perform vital, frequently used functions. And, best of all, source code for these routines is provided so that they are not only useful as they are, but also stand as excellent assembler code examples.

CAPE68k Features:

- ARexx macro support
- Optional optimization
- Generates standalone, binary, or BLinkable
- Source size and number of modules unlimited

InovaTools 2 Functions:

- ARexx: Open ARexx port, close ARexx port, wait for ARexx packet, get packet, send ARexx message, reply to ARexx message, extract arguments
- File: Open, close, read from, write to, detemine size (open and closed), load entire file into allocated memory
- Console device: Open device, close it, print char, line, or dec/hex number
- And: Formatted output, library- , linked list-, and memory- management routines, plus math and extensive string functions

The Amiga and the Inovatronics Assembly Line: The only way to *program!*

Inovatronics, Inc.

The REAL POWER in Power Programming
11311 Stemmons Freeway, Suite 8
Dallas, TX 75229 Phone:214/241-9515

Trademarks: Amiga: Commodore International. CAPE68k, InovaTools, Inovatronics: Inovaronics, Inc.

Volume 2, Issue 2

Publisher
Antony Jacobson

Vice-President Operations
Jeannie Lawrence

Assistant Advertising Manager
Julie Cale

Editors
Nick Sullivan
Chris Zamara

Assistant Editors
Tim Grantham
Malcolm O'Brien

Contributing Writers
Steve Ahlstrom
Ben Blish
Daniel Brookshier
Anthony Bryant
Jim Butterfield
Betty Clay
Don Curtis
Matt Dillon
Andy Finkel
Brad Fowles
Eric Giguère
Cathryn Graham
Eric Graham
Chris Gray
Derek Grime
Jay Gross
Mike Hubbartt
Eugene P. Mortimore
Bryce Nesbitt
Hobie Orris
Rob Peck
Larry Phillips
Dan Schein
Steve Simpson
John Toebes
Victor A. Wagner
Doug Walker
Clark Williams
James D. Williams
Doug Winger
David Wood

Cover Artist
Christine Delezenne

for the AMIGA Transactor
The Magazine for Amiga Programmers

Programming in M2AMIGA 30
by Anthony Bryant
Now you're talking - easy-to-use speech routines in Modula-2.

Getting A Handle On File Systems 34
by John Toebes and Doug Walker
In their first part of a series on file systems, John and Doug provide the most comprehensive docs yet on AmigaDOS packets.

DOS Shells: No More CLI! 42
by Mike Hubbartt
A shell brings extra power and convenience to your CLI operations - but which one is right for you?

MIDI Programming Primer 45
by Daniel Brookshier
How MIDI protocol works, complete with reference for all command sequences.

Checking Stack Usage 49
by Nick Sullivan
How much stack space does your program really need?

Process Spawning 50
by Chris Zamara
Creating a process from a function in your program is certainly handy, but it's never been easy - until now!

Using The Bison/Yacc Parser Generator, Part 2 52
by Eric Giguère
Following up on his theoretical discussion last month, Eric uses Bison to create ECalc, a fully operational expression evaluator with variables and floating point functions.

A Matter Of Style 58
by Ben Blish
Everyone wants to write portable, maintainable code - Ben presents some ideas that will help you do just that.

Departments and Columns

Letters 4

Amiga Dispatches 8
by Don Curtis
Board-swapping overkill, new toys, and more toys coming...

ViewPort 12
by Larry Phillips
What's the ideal bundled language for the Amiga? Larry looks at alternatives to AmigaBasic.

ACCESS - The best of non-commercial software 14
by Steve Ahlstrom
JR-Comm is an impressive new terminal program - Steve takes it out for a spin.

Mastering The Raster 18
by Derek Grime
The Amiga videophile has many tricks at his disposal - but don't use them all at once!

Assembly Language Programming 21
by Jim Butterfield
Jim discusses some Amiga-specific issues: program hunks, startup code, and purity.

News LINK 76

Reviews

Three hard drive backup utilities 64

Interfont 69

DevPac 72

Hisoft BASIC Professional 74

About the cover: *Portrait - Egon* by Christine Delezenne

This month's cover artist, Christine Delezenne, also did the 'Bathers' picture for the cover of Volume 1 Issue 6. Christine studied at an experimental art school in France, where the Amiga was used in a course on computer art. Her work is generally done using Deluxe Paint II, with some 'water-colour' effects added using Digi-Paint for the extra colours available in the HAM graphics mode.

Transactor for the Amiga is published monthly by Croftward Publishing Inc., 10-85 West Wilmot Street, Richmond Hill, Ontario, L4B 1K7. ISSN# 0838-0163. Canadian Second Class Mail Registration No. 7690, Gateway-Mississauga, Ont. USPS Postmasters: send address changes to: Transactor, PO Box 338, Station C, Buffalo, NY, 14209.

Croftward Publishing Inc. is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names are registered trademarks of Commodore Inc.

Subscriptions: see order card at centre of magazine for complete ordering information and prices.

Send all subscriptions to: Transactor Subscriptions Dept. 10-85 West Wilmot Street, Richmond Hill, Ontario, Canada, L4B 1K7, (416) 764-5273. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to our Richmond Hill HQ. For best results, use the postage paid card at the centre of the magazine.

Quantity Orders: In Canada: Ingram Software Ltd., 141 Adesso Drive, Concord, Ontario, L4K 2W7, (416) 738-1700. In the USA: IPD (International Periodical Distributors), 11760-B Sorrento Valley Road, San Diego, California, 92121, (619) 481-5928; ask for Dave Buescher.

Editorial contributions: writer's guides are available by writing to the Richmond Hill address above. Write to the editorial department (same address) with article proposals or articles for consideration.

All material accepted becomes the property of Croftward Publishing Inc., except by special arrangement. All material is copyright by Croftward Publishing Inc. Reproduction in any form without permission is in violation of applicable laws. Write to the Richmond Hill address for a writer's guide.

The opinions expressed in contributed articles are not necessarily those of Croftward Publishing Inc. Although accuracy is a major objective, Croftward Publishing Inc. cannot assume liability for errors in articles or programs. Programs listed in Transactor for the Amiga, and/or appearing on Transactor for the Amiga disks, are copyright by Croftward Publishing Inc. and may not be duplicated or distributed without permission.

Transactor for the Amiga is produced using Amigas and the following products:

Professional Page
Deluxe Paint II
Transcript
Digi-View

Final output by Vellum Print & Graphic Services, Inc., Toronto

Printed in Canada by Bowne of Toronto

LeTteRs

In defence of MS-DOS ports

I have never felt so strongly about an editorial or article that I had to write a letter to the publication to comment on it. The ViewPort column of Volume 1, Issue 3 had that effect. Larry Phillips wants Amiga owners to have vision, imagination and enthusiasm for our computers. Well, these feelings are what inspired this letter in defence of ports of MS-DOS programs to the Amiga.

I don't know about Larry Phillips but I suspect that most readers, like myself, communicate with others using the written or spoken word. I find it difficult to think of a way to input columns of numbers or a letter to the editor other than through a 'text-oriented user interface'.

We welcome correspondence on topics discussed in the magazine. Letters selected for publication may be edited for length.

Please address your letters to the editor to:

Letters
Transactor for the Amiga
10-85 West Wilmot Street
Richmond Hill,
Ontario L4B 1K7

But, perhaps Mr. Phillips doesn't use his Amiga for such lowly, unimaginative purposes. It was hardly IBM's idea to display information on monitors using rows of text.

I don't find it distasteful to tell prospective Amiga users that the powerful, capable software they want and need to make the Amiga a productive tool is available when they buy the computer, not in the distant future. I don't see any trickery in telling the 'unsuspecting business type' that they can not only run the applications they are used to, but can do so with greater power and efficiency.

As for what programmers will want to compete with Ashton-Tate or Lotus... It will be those with vision, who see the Amiga for what it is or can be - those programmers who want to bring out the best the Amiga can give us. It won't be the ones who are giving us the current crop of half-baked productivity software: software with cute and clever interfaces, lots of colours, pulldown menus and pop-up requesters, but little to no functionality. Is it any wonder that Amiga owners long for software like *dBase* and *1-2-3*?

Pick up any of the Amiga-specific magazines and look for reviews of software. Do they all seem the same to you? While there are exceptions, most seem to be written by Larry Phillips zealots: "The Amiga is a great computer, therefore all software for it is also great!" These reviewers are so filled with enthusiasm for the Amiga that they are blind to the deficiencies and the inadequacies of the products they review. The Amiga world is not unique in this. I just find it more annoying, given that prices for Amiga software are in the same range as those in the IBM world, without the software being equal or better in performance. I am quite aware that there are plenty of high-priced turkeys in the land of MS-DOS, but at least they have alternatives.

You ask if we want to enter the 'cutthroat business market' and compete against 'sterile, boring machines' with the Amiga. I say yes! Having the Amiga positioned as a viable business tool with software already established and in use translates directly to the sales of more Amigas. You acknowledged this but missed the point. The more Amigas in use, the more software that can be sold. The more software sold, the more money these software companies will make - money they can turn around and invest back into more software that is even more powerful for applications not even thought of yet.

If having thousands and thousands of Amigas sitting in business offices running *1-2-3* means more new and exciting products for me, then I'm all for it.

I am not really concerned with the emotional state of businessmen when they buy Amigas, whether it is with the 'eagerness he would exhibit when he buys a typewriter' or with the religious fervour Larry Phillips would prefer. As long as Amigas are being purchased and used, we can all look forward to continued expansion of the software and hardware available for it. The business world will keep going to the MS-DOS machines as long as they are the only ones with what it takes to satisfy them. Give the unimaginative men in the three-piece suits the option of going to the Amiga. Then blow their socks off with the additional power and capabilities. Give them back their lost imagination.

Not that Jerry Pournelle or Steve Gibson need me to defend them, but they have influenced a lot more past and future computer purchasers than you ever will. You can hardly accuse Jerry Pournelle of lacking imagination. And sir, you're no Jerry Pournelle.

I enjoy owning my Amiga 1000 for the unique and innovative computer it is. But I am tired of waiting for powerful programs so I can use my Amiga to its fullest. Having the best hardware in the world doesn't mean anything without software to run it. Wake up, Larry, the Amiga is a computer, not a religious icon. Don't try and deify it. The world has had plenty of deities that people no longer pay homage to. An isolationist attitude will not pay off in the long run.

Phillip Wright, San Clemente, CA

Larry Phillips replies:

Communicating with the written or spoken word is indeed part of computing. However, entering text into a word processor or numbers into a spreadsheet is only one part of a program. I am more concerned with how we get to that point in an application, which does not preclude excellence of operation once we arrive there. Are the *dBase* and *1-2-3* methods the best way we have? I certainly hope not.

Perhaps you remember the older BASIC programs that, when prompting for a 'yes

A CSA TURBO

68020

FOR ONLY

$399?

Hard to believe isn't it? CSA broke the price barrier in 32 bit technology. Now you don't have to settle for a far less capable 68000 accelerator, you *can* have affordability, capability, and speed in one easily installed package.

"1000% faster than my stock Amiga™?"

CSA's new 68020 *Midger Racer*™ flies through MaxiPlan™ spreadsheets over 60% faster than the standard Amiga™, and over 50% faster than 68000 based accelerators. With the addition of a 12 MHz 68881 co-processor, the *Midget Racer*™ can crunch 2 Million FLOPS over 1000% faster than a stock Amiga™.

"600% faster than a 68000 accelerator?"

In fact, programs like Sculpt & Animate 3D or 4D™, Draw 2000™, and X-CAD™ have been written to directly access the 68020 & 68881, and may not even *run* with a 68000 based accelerator. And with a 12 MHz 68881, CSA's *Midget Racer*™ is up to 600% faster than a 68000/68881!

"All that for only $399?"

No, there's more! It's also easily installed into ***any*** Amiga, supports an '881/2 up to 33 MHz, and it's available now. Don't settle for second best, call your Amiga dealer or CSA today!

CSA. We set the standards in performance.

CSA

Computer System Associates, Inc.
7564 Trade Street
San Diego, CA 92121
619-566-3911 FAX: 619-566-0581

Trademarks: Amiga: Commodore-Amiga, Inc.; MaxiPlan: Oxxi, Inc.; 68000/020/881/882: Motorola, Inc.; Sculpt & Animate 3D/4D: Byte by Byte, Inc.; Draw 2000: Aegis, Inc.; X-CAD: Taurus, Inc.; Midget Racer: Computer System Associates, Inc.

68000 DISASSEMBLY

OTG Software

ANNOUNCING . . . DSM VERSION 1.0d

DSM is a full-featured disassembler for the Amiga. Check out these features and you'll see why programmers agree, "DSM is the best disassembler currently available for the Amiga, bar none."

- DSM disassembles any Amiga program that does not use overlays.
- DSM produces assembler-ready output that is 100% compatible with the Amiga assembler, assem. Excellent results can also be obtained using the C.A.P.E. assembler from Inovatronics.
- DSM features true text detection.
- Improved disassembly is supported using the built-in expert system mode.

If your local Amiga dealer doesn't carry DSM, send check or money order to:

OTG Software
200 West 7th Street Suite 618
Fort Worth, TX 76102

Price $67.50 US

TX residents add 7.25% sales tax.

TRY BEFORE YOU BUY!

Best selling games, utilities and classics, plus new releases!

- 100's of titles
- Low prices
- Same day shipping
- FREE brochure

YES, WE ACCEPT:

RENT-A-DISC
Frederick Building #219
Huntington, WV 25701
(304) 529-3232

TxEd *Plus*™

The Text Editor for the Amiga™, *Plus* a whole lot more.

New ! V2.0
Supports FFS. **BlitzDisk** — Disk cache, speeds up floppy and hard disk reads up to 2000%.

FastFonts & FunKeys — FF, the text speedup, and FunKeys, the hotkey window manipulator.

New !
Version 1.3 **ARP** — ARP, the enhanced AmigaDOS™ compatible CLI program replacements. New ASH shell, and complete users manual. Save 70 KBytes on your boot disk with smaller, faster, more powerful replacements for the C: commands.

AREXX — Demo version of the AREXX the macro processor.

TxEd *Plus* is a powerful yet simple text editing environment designed to meet all of your text editing needs. It's small, fast, and fully configurable. The AREXX macro language connection makes TxEd *Plus* an ideal foundation for your complete multitasking system.

"As a programming and general-purpose text editor, TxEd Plus sets the standard."
-Warren Block, Sept '88 INFO Magazine

"(TxEd V1.3 is) ...a very good editor and an excellent value."
-Jan&Cliff Kent, Vol. 1 #9 Amazing Computing

"FastFonts / BlitzDisk provides much more than Facc II for a similar price."
-Warren Block, July '88 INFO Magazine

"I found the ARP commands to be smaller, faster, and more powerful than their forerunners."
-Jeff Blume, July '88 AmigaWorld

"Yes, get TxEd if you're editing almost anything on the Amiga."
-Bruce Webster, July '86 BYTE Magazine

"Still the one for me."
-Louis R. Wallace, Nov '88 AmigaWorld

$79.95

Microsmiths, Inc
PO Box 561, Cambridge MA 02140
(617) 354-1224 BIX, PLink: cheath CIS: 76004,1766

MC and Visa accepted. Mass Residents add 5% sales tax.
Amiga and AmigaDOS are trademarks of Commodore-Amiga, Inc

or no' response, required the user to enter a number to indicate the desired choice. "Do you want instructions? Enter 1 for YES, 2 for NO." I think we can agree that this type of user interface was dictated by the primitive nature of early BASIC implementations. Yet even after string manipulation and comparison became commonplace in BASICs, we still saw this sort of thing in programs. Old habits die hard.

As programming languages become more powerful, as hardware advances in speed, ability, and storage size, programmers should take advantage of the new capabilities. I expect it, I hope you expect it, and most computer users expect it, and should not have to put up with the sort of programs and hardware we had back in 1975. While programs ported straight from the MS-DOS world are arguably better than the examples I have given, they are not written to take advantage of the many features the Amiga has that an MS-DOS machine does not.

I am speaking not only of user interfaces, but of the unique features of the Amiga that are built in, available to every Amiga owner as soon as he opens the box. It is the misappropriation of the Amiga's capability that I object to when I see MS-DOS ports.

I edit text - a lot of it, in English and in a number of programming languages. My editor is fast, powerful, and efficient. It is also friendly, allowing me to use menus or keyboard as I see fit, allowing me to use other programs in conjunction with it via the unique interprocess communication abilities of the Amiga. It is decidedly not a port of something like *WordStar*. If it was, I would not have it. It's as simple as that. If a program does not do its job well, or does not have a good user interface, I will cast my vote by keeping my wallet in my pocket.

I do not deify anything. I do admire - make that 'strongly admire' - advanced technology. If anything, the IBM PC and its descendants have been deified by those who place more stock in the 'right' three letters on the front of the case, than in the *contents* of the case. In my opinion, if any other company had tried to foist such outmoded designs driven by such primitive operating systems on the computer-buying public, it would have been laughed off the market, and no software to speak of would have ever been written for it.

No, I'm no Jerry Pournelle, or Steve Gibson, and far be it from me to try to be either one. I am content to let them rave over the latest technology 'advance' from the mainstream business micros, while I attempt to promote the writing of innovative programs that do the necessary jobs.

I think perhaps your views and mine are a lot closer than might be evident from this exchange. We both want powerful software that takes advantage of the Amiga. I just don't think that a program has to look like a direct port in order to perform well.

Larry Phillips, Vancouver, BC

The Power of AMIGA with the Stroke of a Pen

Inkwell Systems introduces the *NEW* 184-A Light Pen with Amiga Light Pen Driver which provides an easy-to-use, natural alternative to the mouse for data entry on the most popular Amiga software programs.

184-A LIGHT PEN

YES IT CAN!

The Light Pen and Amiga Light Pen Driver work alternately with the mouse and keyboard to e-x-p-a-n-d your data entry capabilities.
Got an *EASYL?*
Great!
Install it in the expansion port and combine it with The Light Pen and the mouse for the ultimate in graphics.

- Lightweight
- Triangular shape fits great in either hand
- Two touch switches- no need to touch the pen tip to the screen.

YES IT DOES!

- Supports the Amiga 500, 1000, 2000, and A2000.
- Operates in both interlace, non-interlace and overscan mode.
- Compatible with the most popular Amiga programs for Graphics, Business, and Animation.

YES IT IS!

The Light Pen and Amiga Light Pen Driver is AVAILABLE NOW! ONLY $129.95. Call Inkwell Systems or see your nearest Amiga Software Dealer for a demonstration.

Inkwell Systems
CREATORS OF PENWARE™

1050-R Pioneer Way
El Cajon, CA 92020
619/440-7666
FAX: 619/440-8048

Amiga Dispatches

A narrow escape from board-swap overkill

by Don Curtis
Copyright © 1989 Donald A. Curtis

Don Curtis is a police officer in Denver, Colorado, who has for the last three years been assigned to program design and development along with system design and maintenance of ten AT & T Unix 3B2 computers. In his spare time, Don is an assistant sysop on CompuServe's Amiga forums.

You know, I never did want to become a 'guru' on the BridgeBoard, but it seems that problems and/or new things keep cropping up relative to the Bridge all the time... so here's some more:

My BridgeBoard died a week ago. It had been fairly flaky for a long time. It seemed to have a mind of its own when it came to booting. Sometimes it would boot just fine; at other times it would hang or guru the machine. It also seemed extremely sensitive to positioning in the slot. Any deviation from 'just right' and it wouldn't work.

Well, last week, it decided it was just going to quit - it wouldn't work no matter what I did. It wouldn't access the drives, and the *binddrivers* command would simply hang, either locking the Amiga or giving me a guru.

For those of you thinking I had done some damage while cleaning the machine (see last month's Amiga Dispatches for more on that subject)... nope, the card seemed to work fine for a few weeks after that. In fact, it had seemed to work better than before; I'd had no problems at all after the cleaning... until this.

In any event, I tried all my little tricks to get things working again: reseating the chips and board(s), reloading the software, and so on. That didn't work, so off it went to the dealer's for a repair.

This is where things got interesting. The service tech told me that CBM supplied him with a diagnostic disk for the Bridge, but that to use the disk... the Bridge had to be booted. Talk about a Catch-22: you can't diagnose problems unless you can boot the Bridge; if the Bridge is broken, it won't boot. Commodore's answer to that problem is to send the Bridge in for a swap at $235!

Now, Roger (the service tech) and I are smart enough to know that CBM has to have some form of test jig for the Bridge. He made several calls to CBM and found out that, in fact, there *is* such a test jig and program available *in Europe* but not here. It seems the folks in Europe won't stand for board swaps - they want chip-level diagnostics, so Commodore Europe was able to provide them with what they wanted. The Commodore folks here in the US don't feel the same way. They figure we'll stand for board swaps, and don't provide board-level diagnostics for the repair shops. They said that the board-level stuff wasn't 'just right' (still had some bugs), but that at some time in the future they would release the debugged board level diagnostics (and test jigs where necessary).

Let me digress from the specific subject of the BridgeBoard repair to repairs in general for a minute. In some cases, a board swap is the right way to go; in other cases, it isn't. Service tech time runs in the $40 to $60 per hour range, depending on where you are and how good the tech is. Some problems - intermittent ones, in particular - are extremely hard to diagnose. Most of the chips in the computer are soldered in place, in order to lower production costs and increase overall system reliability. You don't have to worry about loose chips, but you *do* have to worry about repair costs.

An example of the repair problems that can cause (and the expense) is a situation where you attempt to trace a signal. The signal is fine going into chip 'A', but disappears on the output from chip 'A' to chip 'B'. That could mean that chip 'A' is bad, having no output, or that chip 'B' is bad, shorting the input. With socketed chips, you simply pull (or replace) chip 'B'. If the signal returns, 'B' is bad; if it doesn't, 'A' is bad.

With chips soldered in place, it's not so easy. With multi-layered boards, you have to be very careful when removing a chip. It's quite easy to destroy a multi-layered board with improper soldering techniques. It's also not unusual that the chip being removed gets ruined. With a $3 chip, that's no big deal; with a $50 (or more) chip, it *is* a big deal.

So, the tech has to make an educated guess as to whether 'A' or 'B' is bad and replace it. If the tech is wrong, then the other

chip has to be replaced also. All that takes time (labour cost) plus the parts cost, which can be rather high. In cases like that, where you can't quickly isolate the problem and fix it, a board swap is probably a good idea - it's less expensive for the customer in the long run. Not only that, but if the tech takes three or four hours per machine isolating a problem, pretty soon there'll be a pile of machines awaiting repair, and a lot of customers upset at having to wait a week or two before their machines are even looked at.

There are also cases, however, where a board swap is ridiculous. One of the members on CompuServe reported that his A1000 went bad. He took it into the shop and they told him that he needed a new motherboard. That cost him $180 plus the labour to replace it. When he asked what was wrong with his motherboard, he was told, "One of the memory chips was bad." Now, if that wasn't just a 'standard line' that the shop feeds customers when they ask what was wrong, then he was billed $200 or more because a $10 chip went bad. That's garbage! If they were able to figure out that it was a single memory chip, then they should have been able to isolate that chip and replace it for one heck of a lot less than $200.

Because Commodore won't supply the proper diagnostics, though, many shops don't hire fully qualified techs. They hire folks who have minimal diagnostic skills but are able to swap socketed chips or whole boards. They have turned from repair shops into 'board swap' shops. Their so-called techs do minimal checks of the chips they can replace and, if those aren't bad, they send the board in for a swap.

If Commodore ever does get their diagnostic programs 'just right', then perhaps we can have real repair shops again - shops where the tech would know which problems really needed a full board swap, and which problems could be fixed fairly easily in the shop.

Right now, I think Commodore is doing the Amiga community a real disservice by not providing the techs with the proper tools. Even tools that aren't fully debugged would be better than no tools at all, in my opinion. As long as the limits of the current diagnostic software are known, a qualified tech should be able to work around them.

Anyway, those tools aren't currently available here in the US, so back to my BridgeBoard problem. Before we decided to send my Bridge in for a swap, Roger decided to try to figure out the problem. He had a working Bridge in his shop and, from talking with Commodore, had an idea where the problem might be. At the very least, he could try swapping some of the chips that are socketed with those on the working Bridge to see if he could isolate the problem to a specific chip. As he pulled the chips, he bent the pins of the chip carrier out a bit to ensure good contact when the chip was re-inserted (these were the large square chip carriers). Of course, along the way, he also had to reseat the board a few times because, no matter what, it's still got to be seated right and it's not the easiest board to seat.

Suddenly the board began to act as though it were making up its mind to start working: it would access the drive when it hit the *binddrivers* command, but the drive wouldn't reset. Roger said he'd seen that problem on the 5.25" drives that Commodore supplies with the Bridge before. Sometimes those drives seem to simply hang on track 0 - they won't seek and won't reset even with a full power off/on cycle. He had a very simple fix for that. You power off and move the head on the drive up a bit so it's off its parked position. That seems to 'wake up' the drive next time you turn it on. By the way, this is the 5.25" floppy drive we're discussing here - do *not* move the heads on a hard drive!

Well, darned if that didn't do it. We turned the Amiga back on and the Bridge booted just fine now that the drive remembered it was alive. Back to Catch-22: there was no need to run diagnostics now, the board was working fine. Well, now we had to put the original chips back in one at a time to see when it broke again... and hopefully that would be the chip that needed replacement.

Yup, you guessed it, all of the original chips went back in and the board continued to work just fine. Apparently the problem was a bent pin in one of the chip carriers (probably the Bus Translator chip). It just wasn't making good contact and normal oxidation of the metal finally caused it to just quit making electrical contact at all. An hour's worth of labour, and my Bridge was fixed. I brought it home, put it back in my Amiga, and it's been working just fine for several days now. In fact, it's working better than ever: no more positioning tricks - it slipped in and started working right away. I did have to do that same little 'move the head' trick on my 5.25" drive the first time. Since then, though, it's been fine - the head locking wasn't the original problem. Even though the heads were locked, the LED on the drive would come on, and the bridge would boot to the point where it attempted to look for the DOS disk in drive A:. When I first brought my Bridge in, it wouldn't even pass *binddrivers* and the drive LED would not come on at all. I suspect that when the Bridge failed, it somehow or other caused the heads to lock in position. I don't know if that's really possible, but it's interesting that it occurred on two machines when using my Bridge while it was defective.

I ended up paying about one fifth what a board swap would have cost me. So what's the moral of this story? If your Amiga ever breaks, try to find a shop where they'll at least be willing to try a few things before they decide you need to spend several hundred dollars on a board swap. A competent tech can do some fairly quick tests on his own to see if the problem can be easily isolated and fixed. A good tech can usually tell, fairly quickly, if the cheapest route for you is a board swap or a shop repair. A good tech will also do the little things like resetting the pins on a chip carrier, just to make sure the problem is the chip and not the carrier (as my case turned out to be).

It might be useful for you to go down to your dealer's and have a chat with the tech. Even if your Amiga is working fine,

you'll at least get some idea of how that particular shop works. If they are just a board-swap shop, you might want to look around for another repair shop. Hopefully, your Amiga will never break but, if it does, you might save yourself some time and money if you know of a good repair shop beforehand. And while you're looking around, you might want to see if the tech can try to convince Commodore to release what diagnostic software they have now.

New toys

While I was at my dealer's, I had a chance to look at the new A2286 AT-compatible Bridge Board. They'd got three of them the day before, the first to arrive in Denver. It's a nice unit, but rather pricy at $1499 US list. The A2286 is an 8Mhz, 1 Meg RAM, AT-compatible clone on a card-daughterboard combination. It takes up two Amiga slots, one of which has to be a Bridge slot. That means that if you put it in the left Bridge slot, you'll no longer have a fully AT-compatible slot left. However, if you wish, you can have the extra two 36-pin card-edge connectors installed to make all the MS-DOS slots fully AT-compatible. The holes, along with the circuit traces, are already on the A2000's motherboard. The connectors simply need to be soldered in place. If you place the A2286 in the right Bridge slot, you'll use up one of the Amiga-only slots also; however, you'll still have one fully AT-compatible slot available with no motherboard modification needed.

The A2286 comes with a 1.2 Meg 5.25" floppy, MS-DOS 3.3 and the new Janus release 2.0 software. It supports MDA and CGA video modes through the Amiga's monitor (just like the A2088 Bridge). There is also a socket on the board for an 80287 math chip if you wish to add one. The board looks well built (no last minute wire add-ons), but I do see one glaring omission: unlike the A2088, this board has *no* external drive connector! If you want to add a second floppy, you'll have to provide a drive, power supply and case, and then run the ribbon cables into the back of the A2000 through an open slot hole. Finding an open slot hole is no problem - you can remove the cover from the slot to the right of where you've added the A2286's main board. That slot is taken by the A2286's daughter board, so it can't be used for an expansion board anyway. To be honest, though, that's going to look sloppy. Also, since it doesn't have the db23 connector, you can't directly use an A1020 (Commodore's 5.25" Amiga add-on drive) with the A2286.

Unfortunately, my dealer hadn't installed the card in a unit yet for demo purposes, so I couldn't test its functionality. However, I could test the new Janus release 2.0 software, because the same software works with the A2088 and the SideCar. All the software is completely changed, and some new things have been added.

You can now use the A2000's clock to set the date/time on the MS-DOS side. You can use the Amiga's mouse as a MicroSoft bus mouse. You can now autoboot the Bridge from an AmigaDOS file used as a real (not just a virtual) drive. In fact, this drive is so 'real' to the Bridge side, you need to use FDISK to partition it and FORMAT to format it before you can access it. Not only that, but this time, when you create it - using the Amiga program MAKEAB - it gets created full-sized properly. JLINK (which is still available) wouldn't create its virtual drive full-sized, so you had to fill it first to prevent errors if you forgot to unlink the drive (see Amiga Dispatches in *TransAmi*, volume 1, issue 1, for more information on this subject). AREAD and AWRITE have additional options beyond the **/b** switch, and this time they are documented. AREAD and AWRITE do both character conversion and linefeed to carriage return/linefeed conversions by default during their operation. The new switches allow you to turn one or the other conversion off.

And after all that mess we went through a few issues ago getting FFS to run on a JANUS drive, DJMount now fully supports FFS on a JANUS drive. Also included is a new Amiga FORMAT command that works on a JANUS drive under OFS or FFS. By the way, one more little quirk popped up using the old JANUS software and FFS on a JANUS drive. A few folks tried it using the first partition on the MS-DOS hard drive as the JANUS drive and it didn't work at all. They moved the JANUS slice to the second partition and it worked fine.

One other thing that's nice about the new software is that if your Bridge fails to boot, you'll no longer get a 'Task Held' requester and then a guru when you cancel the requester. Some of the directories on the Amiga side have also been moved around, and you can't mix versions of the software. So if you intend to upgrade, make sure you replace all the necessary files, and that they go in the proper spot. Also, make sure you run PCPrefs and set the RAM location for the Janus load properly before you try to use the Bridge. For the SideCar and A2088, the proper RAM location is E000, while the A2286 requires its RAM location to be D000.

Commodore has not yet come up with a policy on how to upgrade from the original JANUS software to the new release 2.0 software. It's not known if dealers will be able to make a copy of the disks for folks who have the current software, or if they'll have to buy it as a separate package. If you want to upgrade, get in touch with your dealer; by the time you read this, a policy may have been set.

And more toys coming...

AMIX (Amiga UNIX™) and the A2500ux have been described in a 'Technology Preview' bulletin from Commodore. A Technology Preview is *not* an announcement of a product's availability, nor is it a final specification. It is merely an announcement of a product as it currently stands and is expected to become available at some time in the future. The bulletin states the following about AMIX and the A2500ux:

The A2500ux will come with one megabyte of 16-bit RAM and four megabytes of 32-bit RAM on the A2620 card. An 80-Meg hard drive with the A2090a controller is included, along with a

150-Meg streaming cartridge tape-drive. AMIX will be fully compatible with AT&T UNIX System V, Release 3.1. It will be able to address one gigabyte of virtual address space with paged memory management. Software support for multiple users, multiple video screens, streams and remote networked filesystem (RFS) is included. A proprietary windowing interface is included. Other software included with AMIX will be an optimizing C compiler with standard development tools such as *cc*, *sdb*, *make*, *yacc* and *lint*. The standard text utilities will also be included, such as *email*, *uucp*, *nroff*, *vi*, *man* and *spell*.

The A2500ux with a beta version of AMIX is available now to qualified members of the Amiga Commercial Developer Support Program.

Another Technology Preview bulletin talks about the High Resolution Color Graphics Card. This card will support resolutions up to 1024 by 1024 with eight bit-planes providing 256 colours (from a palette of 16 million colours) plus two overlay bit-planes providing three additional colours simultaneously.

A TI TMS34010 high speed graphics system processor will execute graphics at up to six million instructions per second. A standard TIGA graphics library provides a library of high-speed graphics commands for the TMS34010.

The PVA2350 Professional Video Adapter is described in yet another Technology Preview bulletin. The PVA 2350 is a frame-grabber operating in real time at 1/60th of a second resolution. It can capture odd, even or both video fields in full colour, and supports all Amiga graphics resolutions including HAM. The card also works as a digitizer and genlock with both RGB and composite outputs.

A final Technology Preview bulletin describes the A590 Hard Disk controller and expansion memory unit. The A590 is a complete 20-Meg hard drive specifically designed for the A500. It will auto-config and auto-boot under v1.3 of the Amgia OS. It operates as a DMA device, attaches to the side port of the A500, and has a Mac-style SCSI connector on the rear to allow additional drive expansion.

The A590 contains sockets for up to two megabytes of expansion RAM. It is my understanding that the A590 will also work on an A1000, although this was not specifically mentioned in the bulletin.

Some of these products were officially announced as products (rather than Technology Previews) for the European markets at the recent CeBIT show held in Hannover, West Germany. That means they should also soon be available in the US market.

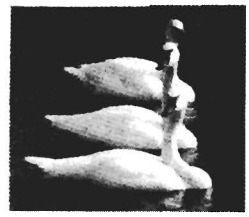
While I griped above about some of Commodore's service policies, I have nothing but good things to say about their Engineering Department. The above previews tell me that Commodore is working hard to make the Amiga a very serious contender in the computer marketplace. That can be only good for those of us who already own an Amiga.

CygnusEd Professional

by CygnusSoft Software. Published by ASDG Inc.

"The New King of Editors,"
68000er, February 1988

"This editor is definitely one of the best I've used on any machine, and I've used many."
The Amiga Sentry, Sept. 1988



CygnusEd Professional brings unparalleled levels of flexibility and speed to the **Amiga.** features include:

Crash Recovery. Variable screen resolution up to 1000 by 800. Full Macro capability. Over 200 functions and variables available through AREXX. Hot Key operation. Screen refresh at over 30,000 cps. Search and replace at over 100,000 cps. Blitter based horizontal and vertical scrolling. Vertical and Horizontal block cut and paste. Many Word Processing features. Many Programmer features. Edit multiple files on screen at same time.

Asynchronous Printer support. Super interprocess communications facilities. DOS and AREXX commands can be launched from inside the editor. User definable "bookmarks". International keyboard and video support. Mouse or keyboard based Turbo Scrolling. Keyboard and mouse can be interchanged for everything except entering text. Will create icons for WorkBench users. Edits binary files. And **MUCH more!**

ASDG INCORPORATED
925 Stewart Street
Madison, WI 53713
(608) 273-6585

Available At Your Local Dealer $99.95 U.S. Dollars

CygnusEd Professional Copyright 1988 By CygnusSoft Software. Amiga and Workbench are trademarks of Commodore-Amiga Inc. Developers are encouraged to contact ASDG for possible publication of your products.

1 MB/min HD BU rate achieved!

ExpressCopy Only $44.95 US

"The Fast, Safe, and Reliable road for HardDisk Backup and Archival."

ExpressCopy can perform hard disk backups and copy files from hard disk to floppy disk at a rate of up to 1MB per minute, 600k Bytes per minute with verify of backup disks. Backup disks have the same directory structure and file attributes of the original files, and can be used normally. Either Normal or Fast FileSystem compatible disks can be written. Files to be copied can be selected by their DateStamp, filename pattern matching, the Archive Bit, and by source directory. Options for setting the Archive bit, verifying the data written to floppy disk, and estimating the number of disks required for the backup. Up to 4 copies of the backup disks can be created at a time, or disks can be pre-loaded in up to 4 disk drives. New disks are automatically formatted and verified. If a bad disk is found during verify, you can insert a different disk and restart the backup from the beginning of that disk. No need to completely restart the backup. Parameters to be used for backups can be saved in configuration files.

Designed for effective multitasking. Change ExpressCopy's priority while it's running. You can HALT the backup and use the Disk Drives normally, then RESUME the backup.

Backup restoration can be done using any file copy program, your favorite Directory Utility, or by ExpressCopy's Restore program.

ExpressCopy has no copy protection, and can be used from both the CLI and Workbench.

Express-Way Software, Inc.
PO Box 10290, Columbia, MO 65205-4005
(314) 474-2984

Workbench is a trademark of Commodore-Amiga,Inc.

ViewPort

Out with AmigaBasic, in with...

by Larry Phillips
Copyright © Larry Phillips 1989

Larry Phillips is an Amiga hardware and software hacker from Vancouver, and a sysop on CompuServe's Amiga forums.

If you ask an Amiga owner about his computer background, the response could be nearly anything. From first time computer owners to those with extensive experience in mainframes, from systems programmer to end user, from beginner to expert, they come from *all* backgrounds. Many are served well by the environment they find on the Amiga, with applications or programming languages. The programming side of the Amiga owner base has many tools to call upon, including assemblers, or compilers for C, Fortran, Lisp, Pascal, Modula-2, APL, and so on. Application programs abound, covering many interests. There is, however, one area that could be improved upon.

Not everyone likes to program, and there are those who shudder at the very thought of writing any kind of program, even simple script or batch files. The Amiga owner in this position must either be content with existing programs, await the development of new applications, or actively convince a programmer that a desired application is worth doing.

At the other end of the programming spectrum are the true programmers, the developers of application software, utilities, games, and more. There are the 'language junkies' who try any language available to them. In fact, some of these people have computers for no other reason than to write programs. These are the people responsible for writing the many programs we find on the Amiga. They are not about to write programs that they don't have a reason to write. The reason may be that they need a particular program, that they see a need for it, or that it will financially benefit them.

Between these two extremes, you will find many Amiga owners, with a full range of ability and desire to create programs. That is where things could be improved upon.

There is a very large number of Amiga owners who would, given the tools, write small, simple programs. They would write the little utilities that nobody else seems to want to write. These could be the 'one-shot' programs, the very specialized programs that might only be useful to one person. They could be more ambitious, growing over time to become full-blown, generally useful programs. Not everyone wants to have to buy a compiler or learn a new language in order to do this.

Imagine the person coming from the C64, Atari 800, IBM PC, or any other small computer, having written a number of programs in BASIC, and wanting to do the same on the Amiga. They open the Amiga box, set everything up, and the first time they want to write that little utility, they come smack up against AmigaBasic. I don't know what the BASIC environment is like on the IBM PC or the Atari (or most other 8-bit micros), but I can tell you that the BASIC environment on the C64 is pure luxury when compared with that of AmigaBasic.

Now those Amiga owners who want to write the occasional program because they need it might put up with the deficiencies in the user interface of AmigaBasic, but what of the person who programs mostly for fun? It sure isn't much fun watching the windows redraw, or realizing right after you run a program that you have a 'syntax error' and must yet again wait for the requester to appear, with its one and only one choice, reach for the mouse, cancel the requester, and wait for the error window to finally go away.

AmigaBasic in itself, as I have said before in ViewPort, is a decent implementation of the BASIC language, but only if you can ignore the bugs and the pitiful user interface. The only reason I mention it again is because I have now seen a BASIC for the Amiga that is easy to use and friendly, and is compatible with AmigaBasic (minus all the bugs, of course).

This BASIC is from MichTron, a company formerly producing software for Atari computers. It's called 'HiSoft Basic Professional', and lives up to its name in more ways than one. HiSoft Basic provides an environment that is pleasing to use, with an integrated editor and error finder, allowing you to edit, compile, correct errors, and test the program without leaving the editor. It also allows you to write code that needs a resident library, or code that is fully standalone. The standalone code is

fairly small, the run-time functions being extracted from the library only if needed, and library-dependent code is *very* small, the run-time functions being provided by a 46K shared library. You are not restricted to their editor and environment, so you can use the editor of your choice and the CLI to write and compile programs. The thick manual is accurate and readable. All in all, it is a very well thought out and implemented package.

It's difficult to talk about HiSoft without doing so in glowing terms. Though I am not really a fan of BASIC, I can appreciate a job well done in any given language implementation. It's also difficult to talk about it without giving it a mini-review, but I wanted to let you know the differences between AmigaBasic and the HiSoft package before going on.

Given the enormous difference in usability between these two BASIC packages, it pains me to see AmigaBasic still being provided with the machine. It pains me to see Microsoft profiting from a half-baked effort. It pains me to think of all the Amiga owners who, if it were not for AmigaBasic, might be writing any number of programs, perhaps to the benefit of the Amiga community in general, and most certainly to their own benefit.

A year or so ago, I asked Microsoft, via their on-line forum on CompuServe, if they planned to upgrade their Amiga product. The answer I got was essentially that I should redirect the question to Commodore. The distinct message, though not explicitly stated, was that they were just not interested. Now in fairness, it might well be that they delivered on their contract with Commodore, that the product was accepted, and that in order to upgrade, Microsoft would require further payment. I can't imagine any other reason for a company to leave a program like AmigaBasic unimproved, with all that implies in company image. Perhaps they don't care that all Amiga owners will know just how bad a Microsoft product can be, or perhaps they don't think there will ever be enough Amiga owners to make a difference.

In thinking about it, I am not even sure that I would want Commodore to pursue the matter, given the alternative of packages like HiSoft Professional, and given Microsoft's past efforts on other machines. I do think, however, that Commodore might do well to consider licensing a package like HiSoft Professional for inclusion with every machine. Consider the advantages to providing a truly good BASIC...

As with the Amiga's Workbench and CLI, the languages included with the machine are highly visible to both Amiga owners and potential owners alike. They serve to many, rightly or wrongly, as an indicator of the usability and power of the machine. Should we have to apologize for any aspect of the Amiga when showing it to someone? I don't think so, and presumably Commodore doesn't either. Yet the language they include with machine, AmigaBasic, is an embarrassment.

Providing a decent BASIC implementation would open up the machine to those who are casual programmers. People coming to the Amiga from 8-bit home computer backgrounds would feel immediately comfortable with the programming tool provided, and would, in all likelihood, use the package, perhaps going on to programming in other languages in time.

One question that comes to mind is: "Are there any *other* candidates for an included language?" The answer to that is a definite yes. ARexx fans will tell you that ARexx serves them as well or better than any BASIC possibly could. COMAL is another alternative, though the Amiga version is as yet unreleased. The problem with both of these, of course, is a lack of familiarity with them on the part of casual programmers. COMAL - which for those who are unfamiliar with it is reminiscent of a cross between BASIC and Pascal - was available for all previous Commodore computers, and for MS-DOS computers, but the number of owners using it were few in comparison with BASIC. ARexx (in the form of Personal REXX), is available on the MS-DOS machines but, again, there are very few actually using it. These languages would have to be *learned*, that is to say, by most users, which would inevitably limit their use. Most former owners of home computers are, however, at least passingly familiar with BASIC.

It is my feeling that Commodore should approach a company like HiSoft with a view to licensing their package for inclusion with every Amiga. Amiga purchasers would love them for it. What say you, Commodore?

✓✓✓✓ UEDIT ✓✓✓✓

Programmable Editor
For Technical Amiga Users

"Definitely the most sophisticated text editor to come along last year . . . easy for beginners . . . and advanced users . . . reviewing UEDIT is like an art critic reviewing a lump of clay. You can mold it to suit your needs . . ."

Commodore August 1988

PURCHASING OPTIONS (Choose one. Foreign add $3.)
A) $47.00 + $3 shipping, **Uedit** with printed user reference manual
B) $44.00, **Uedit** with user reference manual on disk.
C) $25.00, **Uedit** with command language doc on disk.

Uedit has quietly become the editor of choice for bright programmers and professionals in the USA and more than a dozen countries. Recently it won the Gold Medal in Commodore magazine.

Features: Ultra power, capacity, and flexibility. **Uedit** has ARexx interface, many word processing features, optional spelling checker, and a truckload of editing features. How do you describe a program that lets you add new features when you feel like it? Yet you can use it right away "as is". Edit 20, even 100 or more files. Auto-load/edit/save a batch of files while you relax. Every command and feature is customizable, even the online Help and Teach Keys facility!

Rick Stiles, P.O. Box 666, Washington, IN 47501
(812) 254- 4986

ACCESS

The world of non-commercial software

by Steve Ahlstrom
Copyright © 1989 Steven Ahlstrom

As the primary sysop on CompuServe's Amiga forums, and a veteran Amiga programmer, Steve Ahlstrom sees just about every notable public domain and shareware program that gets written, and in this column he tells you about the ones that have recently caught his eye. If you have some information you'd like to share through this column, you can leave a message to Steve on CompuServe (his PPN is 76703,2006), or send the info to us and we'll pass it on.

JR-Comm 0.91, by Jack Radigan

Terminal programs used to be a popular item for shareware authors to write. I am not sure why we are not seeing as many new ones now as in the past. Maybe it is because attention has turned to creating the ultimate directory utility or the ultimate file requester. Or maybe it's because the competition is getting very stiff, especially from Keith Young's Access! program.

However, there has been a lot of talk on the networks recently about the state of Amiga terminal software. Many contend that that no program is up to the level of the communications software available for other machines. It seems that someone with the ability to implement the asked-for features was listening.

There is now a new offering from Jack Radigan, called JR-Comm. This program is still in a 'pre-release' version. As of this writing it is labelled version 0.91. I am not sure what 'pre-release' means. I suppose it means that the program is not yet in a completed state (is any program ever finished?) and the author has made it available as a beta test release.

The public will help pin down the elusive bugs and make suggestions for future enhancements. Even in its incomplete state, JR-Comm is more powerful and feature-laden than most other freely distributable and commercial terminal programs.

Some of JR-Comm's more subtle features are: the ability run from a Workbench window or a one, two, three or four bitplane custom screen; the ability to send a break signal (the length of the break signal is user definable); real time and session time clocks; force a modem hang-up; user-sizable review buffer; and clear screen.

JR-Comm offers a wide variety of file transfer protocols: ASCII, CompuServe's B+ protocol, WXmodem, Xmodem, Xmodem-1K, Ymodem, Ymodem-g and Zmodem. Both Compu Serve's B+ protocol and Zmodem allow for the resumption of a transfer that has been interrupted due to loss of carrier. For use on packet-switched networks, you can 'relax the timing'. This increases the time-out period before a 'NAK' is sent. This is necessary for non-windowed transfer protocols such as Xmodem.

When using Xmodem, Xmodem-1k, or WXmodem, 'auto-chop' can be set. These protocols pad data to the last block of a file transfer. Auto-chop removes this padding from binary files. You also have the option of saving to disk what was received in a failed file transfer. This is necessary for use with the transfer resumption capabilities of CompuServe's B+ protocol and Zmodem, and is a nice feature to use with other protocols as well. Should the file that failed the transfer be in a libraried form (.ARC, .ZOO, etc.) you could more than likely salvage some of the files it contained. The selection of protocols is by far the largest available in a non-commercial program.

JR-Comm is an almost 100% complete IBM ANSI emulator. The default colours are set to the normal IBM palette of black, red, green, yellow, blue, magenta, cyan and white. A 16-colour screen is available for use so that the high-intensity attribute is available for each of the eight IBM colours. The blinking character attribute of the IBM display is rendered as a bold charac-

ter in this emulation. An option for selecting the text and background colours of the display is given, so there is no need to change the colour palette itself. An ibm.font file is included - the IBM character set can be used to complete the IBM ANSI emulation.

The phonebook in JR-Comm is one of the program's strongest features. Phonebooks in other programs will bring in a new set of macros, change colours, and change serial parameters. This is often not enough. When calling a BBS or network, you might need to change some other aspect of the program's settings that are not affected by the phonebook. JR-Comm allows the complete reconfiguration of the system. Every user-configurable item JR-Comm offers is attached to the phonebook entries.

The phonebook offers a unique password feature for each entry. This is a nice feature for those who call a large number of BBSes. It allows a unique password to be generated for each entry in your phone directory. Why would you want to do that? Most people who regularly call a large number of BBSes will use the same password on all of them, just because it is easier to remember one or two passwords. On some BBS systems, the passwords are case sensitive. Even if only one or two different passwords are used for the systems called, you may not remember whether it was upper case, lower case, or mixed case on that system.

Another good reason for allowing JR-Comm to generate the passwords for you is security. Should your password become known, your security is breached on the systems you frequent. In the phonebook, there is a string gadget labelled 'Master password:'. You supply the master password and the program-generated passwords are derived from it. You can specify the length of the generated password and whether it is alpha, numeric, or both.

JR-Comm is capable of multi-dialling any number of directory entries. The number of retries per selected entry, the length of time to wait for carrier detection, and the delay period for restarting the dialling loop are all user selectable. The dialler has a certain amount of intelligence built into it. Should the program receive three 'NO DIALTONE' responses from the modem, the dialler will halt. The entry will be removed from your selected dial list if the modem returns a 'VOICE' response. Should three 'NO CARRIER' responses be received for a given entry, that entry will be removed from the list. When a connection is made, the entry is removed from the selected list. This will allow the entry to be skipped when the dialler becomes active again at the termination of the current call. When a connection is made, all parameters for that phonebook entry will be in effect.

For use in the conference mode of commercial services, JR-Comm offers a 'chat mode'. This is an area of the screen to type into that will not be overwritten by text coming into the main portion of the screen. It is set up so that you can type and not worry about exceeding the maximum line length the service allows. The line will automatically be sent when you reach the length limit of the chat window. The chat mode also offers a history buffer. By using the up and down arrow keys you can redisplay text you have previously sent. When the previously sent line is displayed, all of the normal line editing features are available. The text is then sent to the modem by entering a carriage return.

As with most of the more recent terminal programs, JR-Comm offers F-key macros. You can define up to 20 of them for each phonebook entry. You may load in another set of macro definitions at any time. You cannot, at this time, create a macro that will load in another macro set.

One of the features many people have been looking for in a terminal program is the addition of a log file. A log file is a text file that records the activity of the program. JR-Comm offers this feature. Each time the program makes a connection, an entry is made into the log. It will also log the JR-Comm session start and end times, connection via the dialler (rather than manually typing the number to dial), modem hang-up command, carrier drop, and successful downloads, with the file size, total error count and character per second rate for the file transfer.

JR-Comm has a great deal of potential and, as is, it is more powerful than most currently available communications programs. It is not without its flaws however. Some are bugs, some are design flaws.

Currently, JR-Comm does not support interlace mode. Nor does it support non-standard screen sizes. This is a real annoyance for people using PAL systems or who have increased their standard screen size with a utility such as MoreRows.

The chat window does not handle control characters in a way that is useful for me. Because the use of control characters is fully documented, I would have to consider this a design flaw. To send a control character from the chat window you must type a ^ (caret), then the letter. For a control-P you type ^P and hit RETURN. This is very inconvenient. If there is a large amount of text scrolling across the screen, the normal procedure would be to hit control-S to stop the scroll and control-Q to continue. With JR-Comm's approach, you must find the ^ (caret) key (I'm a touch typist but still must look at the keyboard to find the symbols), type the letter, and hit RETURN. By the time I do this, the text I wanted to see has scrolled off the screen. Not only is the control-P sent but the carriage return also - a further annoyance. In many circumstances I do not want to send a carriage return along with a control character.

A nasty side effect of this approach is that the ^ (caret) alone cannot be sent from the chat window. While this is not a major problem, is does cut down on the characters available for use by all of the ASCII artists out there. I much prefer the approach that Access! takes. All control characters are sent from the chat window. No prefixing with another key is necessary. Just hold the control key down and type the character.

In this version of the program there is a serious bug in the serial routines. When text is output to the screen, all other Amiga functions cease to operate. The mouse freezes, left-Amiga-M fails to function, keystrokes are inhibited - all functionality other than text output is stopped. The events are buffered, but it may take the program 10-15 seconds to act upon them. Mr. Radigan is aware of this problem and will hopefully have it solved in his next release.

Though the IBM font is included to fully utilize the IBM ANSI emulation, I would have much preferred that this font be internal to the program rather than having to be placed in FONTS:. Many programs now are requiring the use of special fonts. I sometimes think that software developers forget that not all users have hard disks, or even two floppy disk drives. They can run out of room to keep a forever-expanding FONTS:.

There is an excellent utility written by Gregg Tavares of Cinemaware that will take a font and generate C code for inclusion into your source code. While this will nominally expand the size of the executable file, it solves many problems: the user no longer has to find room for the needed font(s), and mistakes in the user creation of a font directory and the copying of files to FONTS: are eliminated.

It goes almost without saying that I would very much like to see full ARexx support in a future version of the program. User selectable fonts is another high priority item I would like to see supported.

JR-Comm has the potential to be one of the top two or three communications programs, freely distributable or commercial. Mr. Radigan seems to be very committed to improving and expanding its capability.

In its present form (version 0.91), I would not recommend it for exclusive use, but do use it when you need the protocols it offers. Definitely try future versions as bugs are nailed down and even more functionality is included.

JR-Comm 0.91 is a copyrighted work. It is freely distributable with no shareware fee asked. When JR-Comm reaches a 1.0 release status it will more than likely become shareware or commercial.

JR-Comm is available in CompuServe's AmigaTech Library 4 as JRCOMM.ARC. The author, Jack Radigan, may be contacted by sending CompuServe email to 76545,201 or by calling his BBS, the Atlantic County Amiga BBS, (609) 625-2453.

Notes

I mentioned in a previous column that the CAD software PCLO and PCLOPlus would be entering the shareware arena. While that information was correct at the time, it is no longer so. A new commercial product, Board Master, based upon those products will be entering the marketplace shortly. It will be priced under $100 US.

Amiga Genealogy?

Now Shipping Your Family Tree

THE FIRST GENEALOGY DATABASE FOR THE AMIGA

- Standard Reports • Free Form Memo Area
- Supports Over 10 Million Family Members
- Easy To Learn and Use

YOUR FAMILY TREE

Contact Your Local AMIGA Dealer or Send $49.95 + 6% Sales Tax (PA. Residents Only). $4.00 Shipping and Handling (Within U.S.A.) to:
MICROMASTER, INC.
1289 Brodhead Road, Monaca, PA 15061
PH. (412) 775-3000

AMIGA is a Reg. Trademark of COMMODORE AMIGA

Amiga

500 Computer	
501 512K Expansion	$1000
1084S Stereo Colour Monitor	
1010 3½" external	$159.00
2000 Computer	$1489.00
2000 HD Computer	$2199.00
2500 Computer	$3299.00
2010 3½" internal	$160.00
2058 8MB Expansion	$625.00
2088 Bridgeboard w/5¼" drive	$499.00

Epson

LX-800	$199.00
FX-850	$359.00
FX-1050	$469.00
EX-800	$379.00
LQ-500	$338.00
LQ-850	$539.00
LQ-950	$609.00
LQ-1050	$759.00
LQ-2550	$949.00

Panasonic

KX-P1180	$199.00	KX-P1592	$399.00
KX-P1191	$259.00	KX-P1595	$439.00
KX-P1092i	$319.00	KX-P3131	$310.00
KX-P1524	$539.00	KX-P3151	$479.00

Packard Bell

1200B	$79.00	2400B	$149.00

Hayes compatible external modem

PC Plus

918-337-0266

Everything is in stock, ship the same day
Amiga is the registered trademark of Commodore Business Machines
Authorized Dealer for Amiga, Epson, Panasonic

InterComputing Inc.

2100 N. Hwy 360 Suite 2101
Grand Prairie, TX 75050-1015

In business since 1984; Amiga support since 1985
Member of Dallas Better business Bureau

A SERVICE ORIENTED BUSINESS AT DISCOUNT PRICES
Complete Product List available

Call Toll Free 1-800-622-9177
Customer Service & Order Status: 1-214-988-3500

HARDWARE

Memory

8-Up Memorycard 0K	145.75
8-Up Memory with 2MB	649.75
8-Up Memory with 8MB	22.95.00
Starboard with 0k	239.75
Starboard II with 0.5MB	399.75
Starboard II with 1MB	599.75
Starboard II with 2MB	997.75
Starboard Upper Deck 0k	44.75
A1000 256k Front Panels	199.00
A501 - 512k w/clock	179.75

Drives

FData-10 ext. Drive	139.75
FData-20 ext. TwinDrive	249.75
int. 3.5" Drive A2000 only	145.75
AIR-Drive	169.75
Cal.Access CA-880	169.75
HardFrame SCSI Card	239.75
Starboard SCSI Module	97.75
Quantum 80MB Drive	875.00
Supra 20MB HDisk	649.00
Supra 30MB HDisk	749.00

Video / Graphics

A-Pro Draw 12x12	499.75
Amiga Live A 500	299.75
Amiga Live A 2000	329.75
Amiga Live A 1000	219.75
Color Splitter	85.75
Digi View 3.0 NTSC	139.75
Digi Droid	64.50
Easyl 500	299.75
Easyl 1000	349.00
Easyl 2000	349.00
Flicker Fixer	499.00

Imprint	399.75
Mitsubishi Multisyc Monitors for use with FlickerFixer :	
13" Screen size	549.75
16" Screen size	1349.75
20" Screen size	1895.75
Panasonic 1410 Kit	249.75
Perfect Vision	175.00
Polaroid Palette	1995.00
Super Gen	699.75
X-Specs 3D	97.75

Sound / Midi

ECE-Midi w/ RS-232 thru	44.75
Future Sound Sampler	129.75
Soundscape Sampler	79.50
Perfect Sound A500/2000	59.75
Perfect Sound A1000	59.75
Midi Cable 5ft	8.75
Midi Cable 8ft color coded	11.95
Midi Gold for A2000 internal	69.95

Miscellaneous

Epyx 500XJ	17.50
Mouse Pad	6.95
Modem Cable A500/2000	9.95
Multifunction Card	69.75
Phasar Gun	46.75
Processor Accelerator	149.50
Sony 3.5" DS/DD Disks	18.75
Supra Modem 2400 Bd.	149.75

Flexible Data Systems

FDATA-10 Disk Drive

This external 3.5" Drive is a very inexpensive unit to make the life with your Amiga easier.
The FData-10 is compatible with the original Amiga-drive but is quieter, smaller and has a longer cable (27inch) to connect it to the Amiga.

only **$139.75**

Need more than one drive?

FDATA-20 Disk Drive

is the answer-two drive units in one.

only$249.75

Graphics & Video

Aegis Animator	82.50
Aegis Draw 2000	179.75
Animate 3D	99.75
Animation Multiplane	56.75
Animation Apprentice	199.75
Butcher 2.0	21.75
Calligrapher	87.75
Comic Setter	66.75
Deluxe Paint 2.0	94.75
Deluxe Productions	144.75
Director	44.95
Fantavision	39.95
Flipside	39.75
Forms in Flight II	77.75
Interchange	29.75
Intro CAD 2.0	49.75
Kara Fonts	52.75
Lights, Camera, Action	52.75
Modeler 3D	72.50
Pageflipper Plus F/X	99.75
Photon Paint	66.75
Pixmate	49.75
Pro Board	379.75
Pro Net	379.75
Pro Video Plus	199.75
Sculpt 3D	69.75
Sculpt 4D	399.75
Terrain Disk	19.75
TV Show NTSC/PAL	62.95
TV-Text	62.95
Turbo Silver	129.75
Video Effects 3D	129.75
Video Titler	99.75
Videoscape 3D	119.75
Window Print II	22.75
X-CAD	349.75
Zoetrope	99.75

PROGRAMMING & LANGUAGES

A/C Basic Compiler	129.75
A/C Fortran (AB-Soft)	179.75
Aztec C Developer	199.75
Aztec C Professional	145.75
Aztec Source Level Debug.	59.75
Benchmark Amiga Library	64.75
Benchmark C-Library	64.75
Benchmark IFF Library	64.75
Benchmark Modula-2	134.75
DSM Disassembler	42.75
Lattice 5.0	219.75
Lattice-C ++	395.00
Lint	69.00
MCC-Pascal	79.95
Power Windows 2.5	56.75

SOFTWARE

Music

Audio Master II	69.75
Dr. T's DX-Heaven	99.75
Dr. T's Kawai K1 Libr.	99.75
Dr. T's KCS 1.6	166.75
Dynamic Drums	47.75
Dynamic Studio	129.75
M	154.75
Midi Recording Studio	44.95
Music Mouse	49.75
Sound Quest Casio CZ	84.75
Sound Quest D-110	99.75
Sound Quest D-50	99.75
Sound Quest DX-7	99.75
Sound Quest DX-7 II	117.75
Sound Quest Generic	84.75
Sound Quest MT-32	99.75
Soundsc. Pro Midi Studio	124.75
Synthia	66.75

Business

BEST Management System	299.75
Cygnus ED	69.50
Excellence	189.75
Logistix	99.75
MaxiPlan 500	99.75
Maxi Plan Plus	129.75
Microfiche Filer	69.75
Microfiche Filer II	112.75
Money Mentor 2.0	59.75
Phasar 3.0	66.75
Prowrite 2.0	72.50
Scribble 2.0	57.95
Softwood File II (GS)	79.75
Superbase Personal II	99.75
Superbase Professional	199.75
TexEd Plus	54.75
Word Perfect Library	79.95
Wordperfect	197.75

WIN

Yamaha Synthesizer complete with MIDI Interface & Cables

All orders placed between January 1st 1989 and May 31st 1989 are automatically entered in our drawing. No purchase is necessary; to enter, send a postcard with your Name, Address, Phone number and type of Amiga to: InterComputing, Inc. - Yamaha Drawing
2100 N. Hwy 360, #2101
Grand Prairie, TX 75050

Games

4th and Inches	29.75
Andromeda Mission	27.75
Arazok's Tomb	24.75
Arkanoid	19.75
Balance of Power	33.75
Barbarian	24.75
Battle Chess	31.75
Capone	29.75
Carrier Command	29.95
Centerfold Squares	22.50
Defender of the Crown	35.75
Destroyer	27.75
Ebonstar	29.75
Faery Tale Adventure	29.75
Ferrari	32.75
Final Assault	31.75
Flightsimulator	39.75
Foundation Waste	27.75
Guild of Thieves	29.75
Heroes of the Lance	29.75
Hollywood Poker	24.75
Hybris	27.75
Interceptor	34.75
Into the Eagles Nest	24.75
Jet	32.75
Knight Orc	29.75
Major Motion	27.75
Mindwalker	27.50
Obliterator	29.75
Offshore Warrior	28.75
Pioneer Plague	27.95

Portal	34.75
Ports of Call	29.75
POW	25.75
Q-Ball	19.75
Realm of the Warlock	24.75
Return to Atlantis	34.75
Road Raider	27.75
Roger Rabbit	29.95
Scenery Disk 7/11	19.75
Scenery Disk Europe	19.75
Scenery Disk Japan	19.75
Shanghai	27.75
Silent Service	27.75
Sindbad	33.75
Skychase	27.75
Space Cutter	19.95
Space Quest	32.75
Starglider	29.75
Starglider II	29.75
Street Sports	14.95
Strip Poker	27.75
Strip Poker Data 4 or 5	14.95
Surgeon	29.75
Terror Pods	29.75
Three Stooges	34.75
Torch 2081	19.75
Vampires Empires	29.75
Wintergames	14.95
World Class Leaderboard	29.95
World Games	27.75
Zoom	21.75

Desktop Publishing

Comic Setter	66.75
Font Set I	24.75
Laserscript	29.75
Newsletter Fonts	29.75
PCM-Fonts Vol.1-3	27.50
Page Setter	89.75
Professional Page	239.75
Shakespeare	149.75
Studio Fonts	29.75
Zuma Fonts 1,2,3,4 each	24.75

Utilities

Arexx	32.75
Disk Mechanic	56.75
DOS-2-DOS	42.75
EZ-Backup	32.75
Facc II	19.75
Fine-Print	32.75
Project D	32.75
Quarterback	44.95
Raw Copy	39.95

GVP Products

IMPACT 68030 Card	Call
IMPACT 2000 2Meg/0K	299.75
IMPACT 2000 2Meg/2MB	699.75
IMPACT 2000 0k /80Meg	1199.75
IMPACT 2000 2MB/80Meg	1699.75

Trade up!

We will trade in your Amiga 500, 1000 towards a new Amiga 2000 or 2500! Please call for details.

NEW PRODUCTS

GFA Basic 3.0	99.75
Pagestream	139.75
Fast Trak 20Meg/A500	599.75
Tiny Tiger 20Meg	589.75
30Meg	629.75
40Meg	779.75
50Meg	799.75
80Meg	1199.75
Tool-Box w/Power	219.75
Mini-Meg A500 0.5MB	329.75
1MB	489.75
2MB	789.75
Mini-Meg A1000 0.5MB	339.75
1MB	449.75
2MB	789.75

Micro-Meg 512k/Clock	179.75
Micro-Meg 0k/clock	59.75
Synergy 500	199.75
Overdrive	199.75
Subsystem 500	229.75
Subsystem 1000	279.75
Phoenix A500/1000 22Meg	599.75
32Meg	699.75
48Meg	799.75
80Meg	1299.75
Phoenix A2000 22Meg	549.75
32Meg	749.75
48Meg	749.75
80Meg	1249.75

As always we have the most 'customer friendly' terms: S/H $3.95 in cont. USA; Minimum order is $30.00; MASTERCARD & VISA with NO credit card fee; in Texas add 7% Sales Tax.
We ship to APO addresses, RMA# required on all returns. 18% restocking charge. All prices subject to change without notice.

Mastering The Raster

Bitmaps that bite back

by Derek Grime

Derek Grime is president of Beyond Graphics, a Toronto-based studio that uses the Amiga for a variety of graphics, animation and video applications. Derek can be contacted at Beyond Graphics, P.O. Box 1249, Station F, Toronto, Ontario, Canada, M4Y 2V8

The board room is quiet as the executives sit down to another sales presentation. Today they have been told to expect something out of the ordinary. The resident media specialist is on hand, making some last minute adjustments to a three quarter inch VTR and a large screen monitor. The okay is given, everything is in place. Before the show begins the director of marketing stands to say a few words. She explains to the executives that they are about to be introduced to desktop video. No more flip charts or overheads for this forward thinking corporation, from now on the monthly progress reports will be produced in-house by the audio visual department. On an Amiga.

The lights dim and the show begins. Chromium logos reel over perspective grids, numbers bounce and dance all over the screen and pie charts explode in every direction. Colour cycling text scrolls in everywhere as digitized hands point to the important parts. After a few minutes of computer pyrotechnics the show is over and the lights come back up. There is polite applause from around the table. The head of regional sales quips that the video was just like that 'MTV stuff' and everybody nods and chuckles. The executives file back to their departments secure in the knowledge that their company is 'state of the art' and 'leading edge'. For the rest of the morning they think fondly back to the video they saw - the same way one remembers a fireworks display when it is all over.

Not one of them can recall a damn thing about the contents. Form over substance strikes again.

This month I'm going to discuss bad design. Unfortunately, there's a lot to discuss, so for now we'll just concern ourselves with colour and text and the parts they play in an effective video presentation. Just why is there a glut of ugly design in the computer graphics field? How can it be avoided? How can you recognize it and prevent it from appearing in your own work?

Commodore reminds us that "only Amiga makes it possible and affordable". Some of the problems begin right here. In the past computer graphics were too expensive to be available to the average person. Systems were only owned by large firms and they had to pay top dollar for their artists. Not only did the creative staff have to be illustrators with a degree in graphic design, they also had to have no fear of technology. Older systems demanded that the artist be willing to work in the command line interpreter if that's what it took to get the job done. Today anyone can grab a mouse and create the same variety of full colour graphics in minutes. We have an incredible variety of tools, gadgets and toys on hand to help us get the job done. Unfortunately, the results often fall far short of the desired effect. Usually the problem can be traced to some simple flaws in the graphic design. Let's look at some of the most common of these so you know what traps to avoid in the future.

Palatable palettes

In the Amiga's hold and modify mode we can have all 4096 colours on the screen at one time. The natural tendency for the fledgling artist is to try to use as many as possible in every picture. I'd like to suggest that this is because we Amiga users have somewhat of an inferiority complex as far as the colour palette goes. We know that our 4K of colours puts us ahead of the competition. Then again, we've all seen the workstations with their glorious 16 million colour selections. Even the displays offered by something as inexpensive as a VGA card now give you 256 colours on screen out of millions. But look closely at the contents of the images. The artists who created these works didn't just pull colours from all ends of the spectrum. The choices were made only from the hues that suited the graphic. Having thousands of colours on hand in a program like Deluxe Photolab seems to drive many of us into some kind of colour frenzy. It's as if we were trying to show the rest of the world just what our Amiga can do. That's fine in rare cases, such as an illustration of a rainbow, but it often obscures the pictures under a sea of pigments. Ramping up the red, green and blue values to the top of their ranges is dazzling but not pleasing to the eye. The colours all compete with each other equally, and the viewer's focus is split in too many different directions at once. Better images contain one bright dominant colour and all of the others have to back it up.

There are many varied schools of thought on the use of colour, and they all overlap to a certain degree. None of them deal with colour as it pertains to the red, green and blue colour guns in the back of a cathode ray tube. A program like Deluxe Paint allows us the choice between the RGB or the HSL (Hue, Saturation and Luminance) scales. HSL is a much more natural way of working with colours. Hue represents all the base colours in the same way a prism or a rainbow does. Imagine the hue as a beginner's set of paints. Saturation is the scale showing how pure the colour is. If you took a green hue and pulled the saturation all the way to the top you would get a vivid day-glo green. As you pulled the saturation back down the colour would get duller and muddier just as if you were watering down a can of paint. At the bottom, with all the green removed you would have an unattractive shade of gray. On the other hand, the luminance scale is the light that's illuminating the hue. Putting too much light on the colour overpowers it and it changes to white. Taking the light out of a colour changes it to black.

The spectrum of visual colours is usually displayed as a colour wheel. Colour wheels contain all the hues laid out around a circle. It's the best way to get a feel for how the different hues relate to each other. For example red, oranges and yellows would all be grouped together on one section of the wheel. Think of these groups as a family of colours. Families work together to bring out your chosen dominant hue. If your dominant was a bright red then the colours that form the rest of the image should be duller colours from the same family. Colour wheels can be had for a few dollars at any art supply store.

Try this experiment with your favorite paint program: Do a simple graphic or graduated fill in a range of blues. One end can be fully saturated while the other should be taken down a few steps. When you are happy with it, adjust the palette to a different hue such as a red, keeping the other levels the same as before. The results are still pleasant. Now change one end of the range to another member of the family such as yellow and do a spread between them. This gives you a good base to work from. Whenever I have to do a new illustration, I can call up another family of colours from ones I've previously stored on disk. The ranges are prearranged so that they work in harmony. While building up a library is one way to save time, having good palettes prebuilt right into the software would be the best option. Few programs offer this at the moment, which is surprising considering just how trivial the code would be to write. Maybe we'll see more of this in the future, but for the moment, play with colour to get a good feel for the way the Amiga's display handles it. It's easy and you won't get any paint under your fingernails no matter what you do.

There is also some basic psychology at work here. Colours can have many different meanings. Red is hot, warm and alive. Blue is cold, barren and businesslike. Yellow has been proven to agitate people. Needless to say, you should avoid yellow if you are trying to put the viewer at ease. Many computer graphics use the old tried and true blue background behind white text. It has been done a million times in the past because it works. This colour combination is also pretty boring. Amigas give you an incredible selection of colours, so search for your own blends until you find the choices that perform well together.

Graphics that work

Good animated images have to start with effective still images. Motion choreography can create many problems of its own so this month we'll examine the elements needed to create a good still. Before you begin any assignment, you have to break it down into a series of individual pictures. Traditionally this is done in storyboard format. Storyboards represent your production in a comic strip format. Currently there is no software that makes this easier than the tried and true method of using pencil and paper. Revisions happen constantly at this stage, and you should really create the storyboard with the client at your side. Have the customer initial every frame in the storyboard as it's created. This protects you from unreasonable changes that the client may demand later in the production. Of course, you should also charge for the time spent producing the storyboard.

Each and every frame should be saying just one thing. It's important to keep the amount of information you are trying to convey as uncomplicated as possible. Many times in the past I've had clients ask for stills that contain lines and lines of text. Your job is to explain to them why this doesn't work. In a video production your audience is often over ten feet away from the monitor. Even worse, many of the television sets that are used in corporate board rooms are old and of poor quality. Remember that your Amiga monitor is much better than anything your client has access to.

Also, most corporate buyers were weaned on 35mm slides. Computer-generated slides done with more expensive computer systems sometimes have thousands of lines of resolution.

They retain their integrity when shown on a large screen. On the Amiga we don't have that luxury. Control yourself and keep text to a minimum. If you put more than three lines on the screen your video is going to resemble an eye test and the audience will not bother to strain themselves to decipher it. If the client persists in requesting a dog's breakfast, keep some examples of bad text images on disk. Have your customer look at them from a few steps away. They always change their mind after seeing (or not seeing) for themselves.

Background clutter is another pitfall. Avoid background fills, pictures and grids at all costs if text is the main point. It's fun to fill the screen with neat and wonderful shapes, but that approach is better left to the video games. Simplicity is the key so the real message won't get lost.

The easiest way to get the information across is to begin with a dark background. Use no more than three lines of text and try to keep the lines as short as possible. Use large font sizes in light colours. The first line should contain the dominant point and the luminance value of the text should be boosted a little to make it stand out from the other lines. The supporting sentences should be closer in tone to the background colour yet still be perfectly legible.

Fonts should be carefully chosen. Every job you do will have text in it somewhere, so you must build up a library of fonts that work well on video tape. The standard Workbench fonts don't, so stay away from them. When choosing the style, shy away from the more bizarre fonts. Basic block lettering is the most effective on tape, and variations on Helvetica and Eurostyle are safe bets. Don't mix different styles on one graphic or your image will look like a ransom note. In fact, no more than two or three different styles should be used in the entire video to improve the flow.

Beware of gimmick styles. On the Amiga we have many to choose from, and just like colours they carry along a certain amount of cultural baggage. For example, the Countdown font was introduced in the late sixties. At the time, it was considered to be the ultimate in high tech and resembled a blocky LCD screen. For a few years it was everywhere. Today, less than twenty years later, Countdown is no longer valid. Technology has passed it by and it has been reduced to something that designers use as the punchline to a visual joke. Think of the futuristic boomerang furniture from 'The Jetsons' cartoon show and you get the idea. Styles like Broadway make us think of the twenties, and gothic styles now conjure up images of heavy metal bands. Scripty handwritten fonts are light weight and lose themselves on video. The same is true of many of the 'colourfonts' that are available on the Amiga. What appears visually stunning from your desk chair can be visually confusing from ten feet away.

Shy away from playing tricks with the data. Deluxe Paint lets you dropshadow, bend, emboss, slant, reverse, squeeze or smear your text in seconds. That's just for starters. It is so tempting to apply these effects to everything. Many of us feel that we are giving the customer a little extra for the money. Don't do it. All tricks have their time and place. If the client wants them, they'll request them. If you don't throw everything at them at once you can charge extra for nifty effects later on. When they ask you if you can add a dropshadow, stare intently at the storyboard. Scribble some figures and, after a dramatic pause, explain that a dropshadow *can* be done but for a small extra charge. No one needs to know the shocking truth about how easy it is except for you.

Because you're designing digitally now, use it to your best advantage. Build a clip art collection that contains your best designs. Use these images as templates whenever possible. Cut out the old information and replace it with the new. It's easy after you've done a few productions. I often wonder why no software developers offer programs along these lines. Looking through the collection here, the only examples I could find were on the initial 'Art and Utilities' disk from Electronic Arts. They provide a bar and a pie chart template. Certainly, if the Amiga is going to survive in a media service atmosphere, we can do better than this. Even better would be something that would allow you to punch in the information on the keyboard and then have the software take care of the design and colour choices for you. Importing spreadsheet information would be ideal. Some commercial spreadsheets offer this now in a limited fashion, but a dedicated package would be much more versatile.

Next month we'll look further than text and colours. We'll discuss charts and graphs and get into production staging and motion choreography. Until then, try out some of the techniques from this article. The Amiga gives you more power at your fingertips than any designer had access to ten years ago. Understanding some basic rules of good design will let you wield that power more effectively. Once you know the rules you can start to break them. That's what it's all about.

Assembly Language, Part 3

Hunks, startup code, LINKing, and purity

by Jim Butterfield
Copyright © 1989Jim Butterfield

Jim Butterfield is a Toronto programmer, author and lecturer whose many articles, books, lectures and television programs have made his name a household word among Commodore users throughout the world since the early days of the KIM-1 microcomputer.

"Dear Jim: In part 1, you implied that MOVE from a data register to an address register could be followed by a branch which would be dependent on the data moved. 'Tain't so." Dear Reader: Correct. Any command referencing an address register as its destination will not affect the 'flags' which are tested by the Branch instructions.

As it happens, the branch you are referring to (the one following the DOS library opening) will work correctly on current systems. The flag happened to be correctly set by the OpenLibrary subroutine itself. Try spelling 'dos.library' wrongly, and you'll see the branch at work, skipping the rest of the program logic.

However, that's not rigorously correct coding. I'll try to watch it in the future.

Files: preliminary

We'll be reading a file this time, and putting its contents to the screen. Before we do, however, we should look into the concept of working memory. Our earlier program examples avoided this question; it's time to dig in there.

If you have programmed 'simpler' computers, you may have used the simple method of defining the work space you want, and then using it. That still works on the Amiga, but there are several other options that might do the job better.

We'll need to dig into programming methods, but also into the architecture of Amiga programs. It's a worthwhile trip.

Program structure and hunks

On simple computers, a program comes from storage - disk or tape - into memory. The location in memory is usually known in advance, but the working rule is: the whole program comes into one place.

The Amiga could not have effective multi-tasking if each program insisted on loading to a particular address in memory. There would be no way to resolve the conflicts between demands of various programs. So, as you undoubtedly already know, Amiga programs are able to relocate as they load.

But in the Amiga, programs don't need to load as a single chunk. A single program on disk can be made up of several *hunks*. As the program loads, the hunks split apart; each one finds an independent piece of memory to occupy. The Amiga loader makes sure that all the pieces connect together.

Why hunks, as opposed to a single program unit in memory?

There are several reasons. First, it saves the Amiga the bother of finding a piece of memory big enough to hold the whole program; bits and pieces of memory can be found wherever they are. Secondly, certain kinds of data might call for particular kinds of memory; for example, graphics data within your program might need to go into 'chip' memory, even though the rest of the program can go anywhere convenient. Perhaps most importantly, the hunk system lays the groundwork for future Amiga architectural development.

How do hunks happen? You can force your assembler to break your code into hunks, if you wish. You would normally do this if you wished to develop and assemble your code in separate pieces, and have the linker put them together. Even then, you can ask most linkers to 'coalesce' several small hunks together into a single hunk.

Hunk History, Hunk Types

For reasons that seem mostly historical, there are three types of hunk: *code*, *data*, and *bss*. This last stands for the near-meaningless 'Base of Stack Segment'; many years ago, I worked on a system which used a similar *bss* designation to mean 'Block Storage Size'.

The idea was this: program instructions go into the *code* hunk; predefined 'fixed' data go into the *data* hunk; and although the *bss* hunk contained no values (only a size definition), it would set aside memory for your working values. To follow the idea a little further: your program was not expected to modify the contents of the code or data hunks; all working values went to the bss area. You could disassemble a code hunk and find nothing but program instructions there; you could perform a hexadecimal dump of a data hunk and see only data, not code.

These rules were never precisely observed. Some compilers dutifully produce the three hunks, following which the linker might coalesce some of them or even discard 'empty' hunks.

Data doesn't need to go into data hunks. Putting data into code hunks - especially strings such as 'dos.library' - allows the information to be accessed by means of (PC) addressing, with corresponding savings in memory and time. Working memory - the equivalent of bss areas - could be used within any type of hunk. But as we'll see, there's a tendency for working memory to come from completely outside the program itself.

The usage rules for the three types has almost disappeared. For small programs in particular, everything could go into a single code hunk: program, data, and work space. Our example here will do that... but we'll use it to study the work space question more closely.

Work space and purity

Most programs need a place to store data. This might contain such things as variables, pointers, or data buffers. There are at least four ways we can get that work space:

1. Use a BSS hunk to provide the memory space;
2. Define space within our program, perhaps in a code or data section;
3. Ask the system stack to give us space (using LINK and UNLK);
4. Ask the system to allocate a block of memory (using *AllocMem* and *FreeMem*).

With the first two methods, the space is within our program. The last two involve external space: we ask for it, and are granted the memory workspace.

Does it matter which style we use? Yes, if we want the code to be *pure*. 'Pure' code will be able to run more than one job at a time; if we wish to make this program *resident*, the coding must be pure.

Here's the idea: we're about to write a program that will dump a file to the screen. To do this, the program will need to bring data from the file into a buffer. Taking the idea further: suppose we wanted the same program to handle two files at once. With a single buffer, we'd have a mess: one file would overwrite and destroy the buffer contents, and the work would get badly mixed up.

Options 3 and 4 above, however, allow the program to ask for two buffers. That way, the two files could be kept separate.

Program task

Our program will be called DUMP. when the user commands, from the CLI, 'dump filename', the contents of that file will print to the screen. Unprintable characters will be changed to dots, except for newline, which will behave in the usual way.

One more thing: since the file might be very large and dull, we want to be able to stop the file dump. If we detect a CTRL-C key combination the job will terminate.

The first version of this program will *not* be 'pure'. Let's just get the job done.

Startup code

Let's insert comment lines with name and date so that we can remember something about this program when we look at it in the future. Our data will be sparse here to save magazine space. The identity line is followed by system XREF definitions; you've seen most of those before.

```
;   Program Dump - Jim Butterfield. March 15/89.
; Exec library calls
   xref _LVOOpenLibrary        ;-$228
   ref _LVOCloseLibrary        ;-$19E
   ref _LVOSetSignal           ;-$132
; DOS library calls
   ref _LVOOpen                ;-$1E
   ref _LVOClose               ;-$24
   ref _LVOOutput              ;-$3C
   ref _LVOWrite               ;-$30
   ref _LVORead                ;-$2A
```

Keep in mind that we have a 'command tail', whose length is in register D0 and address is in A0. We must save this information... for the moment, we'll copy it to A4 and D4.

```
;-- Initial setup:
Startup  move.l a0,a4;Remember ptr to argument line.
         move.l d0,d4;Remember len of argument line.
```

Assemblers do a lot of 'thinking' for us. The first MOVE command transfers information to an address register. Formally speaking, it should be a MOVEA command. But the assembler figures that out and makes the adjustment.

Now, we'll open the DOS library in the usual way.

```
   lea     dosname(pc),a1      ; name 'dos.library'.
   clr.l   d0                  ; Any version (0)
   move.l  $4,a6               ; using Exec library,
   jsr     _LVOOpenLibrary(a6);   Open DOS.
   move.l  d0,a6               ; Remember Dos pointer.
   beq.s   StartupQuit         ; Exit if error
```

If we did not take the above branch, we have opened DOS correctly and can now go to the next step. We'll call this as a subroutine, using BSR (branch subroutine). We know that the routine will be close by, so we'll tell the assembler that we expect a short (.s) branch.

```
   bsr.s  DOSinit
```

When we come back from that subroutine, the job is done. The program will close the DOS library and terminate.

```
            move.l a6,a1                ; DosBase
            move.l $4,a6                ; Use Exec,
            bsr _LVOCloseLibrary(a6)    ;  close DOS
StartupQuit rts                         ; End Program
```

Here's our first level subroutine. It's not our main job yet. We'll obtain our output handle that will allow us to write out results. Then we'll put a binary zero character at the end of the 'command tail', which probably has a NewLine character in that position. We need that binary zero to create a C-language type of string; that's the kind that DOS will want for the file name we give it.

```
;
;-- Get CLI outhandle:
DOSinit  jsr   _LVOOutput(a6) ;get CLI outhandle,
         move.l d0,a5         ; & then remember it.
;
;-- Change arg file name to 'C string':
checklen  move.b #0,-$1(a4,d4.W)
; binary zero at end
```

The above is quite a combination of address elements. Start with A4, the address of the command tail. Now add D4, the length of the command tail - we can use .W, word length, since we may be sure that the command will not contain thousands of characters! The sum of these two registers will take us one beyond the position we want to go, so we back up with a displacement of -1.

There should not normally be any spaces at the start of our command tail, but we'll scan to remove them just in case. We use the 'indirect with post-increment' addressing mode.

Indirect means that we use the contents of A4 as an address; *postincrement* means that we add one to the address after we have used it. When we're finished, we back up one position.

```
skipspc   cmp.b  #$20,(a4)+
          beq.s  skipspc
          subq.l #1,a4
```

Now we are ready to open the file. The reference book shows command *Open* as:

```
   file = Open(name, accessMode)
    D0          D1        D2
```

In other words, we supply a pointer to the file name in register D1, an 'access mode' value in D2, and then call DOS function *Open*. We will get back a file handle, in register D0.

If you have an assembler with *include* files, the access mode numbers are defined symbolically in file *dos.i*. For the moment, all we need to know is that 1005 decimal (MODE_OLDFILE) will specify that we want to open the file for reading.

```
   move.l a4,d1          ; filename pointer
   move.l #1005,d2       ; MODE_OLDFILE (for reading)
   jsr    _LVOOpen(A6)
   move.l d0,d6          ; file inhandle
   beq.s  DOSquit        ; no good, quit
```

If we do not take the above branch, the file is open and we're in business. Now we will call our main job with another BSR (branch subroutine) call.

```
   bsr.s  DumpFile          ; the main job
```

When we return, we must tidy up. That means that we must close the file that we previously opened. *Close* requires only the file handle in register D1.

```
          move.l d6,d1           ; use the handle..
          jsr    _LVOClose(A6)   ; to close the file
DOSquit   rts                    ;   exit program.
```

We're finally on our way. DOS is active (A6 contains the library address), the file is open (its handle is in D6), we have an output path (handle in A5) and we'll be ready to read data after a little bookkeeping.

We don't want text to run beyond the right hand margin of the screen. Let's count characters in register D7 so that we can trigger a NewLine if the count goes too high.

```
;
; DOS is open and we have our in/out handles. Let's go!
DumpFile    moveq  #0,d7        ; zero column count
```

Here's where we read a block of data from the file. The *Read* command is defined as follows:

```
   actualLength = Read(file,buffer,length)
     D0                 D1   D2      D3
```

So we put the file handle into register D1, the address of our data buffer into register D2, the size of the buffer into D3, and call DOS function *Read*. When it returns, we'll have the number of bytes actually read stored in D0. If this value is less than D3, we're at the last block.

During the next section of code, we'll refer to the address of the buffer several times, as well as the address of the output buffer. To save code, we'll hold those addresses in registers A4 and A3 respectively. Let's read some data.

```
ReadBlock  move.l  d0, d1          ;input file handle
           lea     BufAdr(pc),a4 ;input buffer address
           lea     OutBuf(pc),a3
           move.l  a4,d2          ;copy to D2 for read
           moveq   #BufSize,d3    ;input buffer size
           jsr     _LVORead(A6)   ;so read it
           move.l  d0,d5          ;size of data
```

We have data. It's in the buffer (whose address is held in A4). Time to look at it, count, and output. Register D4 will walk through the buffer area; we won't let it pass the value in D5, which is the end of the buffer. After the comparison, the BCC (branch carry clear) command acts as a 'branch greater than or equal to'. If you know the 6502, this might seem to be backwards; the carry flag is handled differently in the 68000 world. Here's where we scan and output

```
           moveq  #0,d4 ; start of buffer pointer
; Character loop; are we at end of buffer?
NextChar   cmp.l  d5,d4
           bcc.s EndBuf
; We have data!  Let's crunch it.
           move.b $0(a4,d4),d0
```

Register D0, lowest byte, now holds the character from the buffer. We'll bump the D4 pointer and then start to look at the character: is it a NewLine? Is it non-printing?

```
    addq.l #1,d4
    cmp.b  #$0a,d0
    beq.s  NuLiner
```

If we found character 10 (hex 0A), that's a NewLine and we may skip ahead and zero our line character count. Note that the compare (cmp) command is 'byte sized'. Again, to be formally correct, the command should be CMPI; but the assembler will take care of that for us.

```
    cmp.b  #$1F,d0
    bge.s  NotDot
```

We compare the character to number 31, hex 1F. Note that *bhi* (Branch High) will branch if D0 is greater than 31. It's a signed comparison, and we need to talk for a moment about that.

Signed numbers

When your cassette or video player shows 9999, you understand it to (usually) mean a true value of -1. The concept is this: very high numbers can be taken as being negative values if you make the rules work right.

If we ask the CPU to treat a value as *signed*, as opposed to *unsigned*, we do exactly this. The rule works this way: if the high bit of the number in question is *set*, we will consider the entire number negative. If we choose to use this rule, one-byte values don't run from 0 to 255; instead, they go from -128 to +127. The computer handles the numbers in the same way, but comparisons are done in a different manner for signed numbers.

Consider our situation: we wish to reject any character with an ASCII value less than 32 decimal. Those are non-printing characters, of course. Consider: the normal ASCII character set stops at number 127. What are those others with higher values? Some are control characters, and others make up the 'alternate' character set. A program such as DUMP really doesn't want these extra characters printed.

We could eliminate these extra characters (change them to a dot) by a second test; but we have a better choice. If we choose to consider these characters as if they were 'signed', all these 128-and-over characters become 'negative'. A signed comparison will throw them out immediately.

In fact, it's not the comparison that is signed; it's the way we test following the comparison. And *bge* does just what we want: special characters above the normal ASCII range will not be printed. We skip over the next ('dot-making') instruction.

It takes a while to get used to which way round the comparisons go. If we do not have a character of 32 or over, we change it to a dot (hex 2e, decimal 46).

```
    move.b #$2e,d0
```

Whether we have changed it or not, we print (*Write*) the character and increase the column count by one. Remember that we've put the address of the tiny output buffer in register a3, which saves us some work here.

```
NotDot move.b d0,(a3)
    move.l a5,d1              ;file handle
    move.l a3,d2              ;outbuffer
    moveq  #1,d3              ;length
    jsr    _LVOWrite(A6)
; count and send NewLine if needed
    addq.b #1,d7              ;bump column count
    cmp.b  #64,d7             ; at limit?
    bls.s  NextChar
```

If the number of characters has gone over 64 (a value I picked for no special reason), we will not take the above branch (BLS for Branch Less than). Instead, we'll drop through to NuLine and start a fresh line. We also come here if we see a NewLine character in our file.

```
NuLiner    move.b #$0a,(a3)
           move.l a5,d1        ;file handle
           move.l a3,d2        ;buffer
           moveq  #1,d3        ;length
           jsr    _LVOWrite(A6)
```

In a moment, we'll loop back to get the next character and continue our file dump. But first, a message from CTRL-C... maybe.

We detect CTRL-C by calling an Exec function, *SetSignal*. It's not documented too well in the ROM Kernal manual, but the following code does the trick. First, keep in mind that we are about to call the Exec library and thus will need a pointer to ExecBase in register A6. We'd better save our DOSBase pointer by tucking it into register D7.

```
move.l   a6,d7              ;stash DOSbase
moveq    #0,d0
move.l   #$1000,d1
movea.l  $4,a6
jsr      _LVOSetSignal(A6) ;test CTRL-C
move.l   d7,a6              ;restore DOSBase
```

We set up our CTRL-C mask in D1 and call *SetSignal*, then restore the DOSBase pointer as quickly as we can. Now, let's analyze what results have come back in register D0.

We extract a single bit using the AND operation. If you don't know AND, think of it as 'remove all the bits except the one I'm interested in'.

```
andi.l #$1000,d0
tst.l  d0
beq.s  NotCtrlC
```

If D0 contained zero after the AND masking, we can breathe a sigh of relief and go back to our file reading. If not, we've detected CTRL-C and should stop the program.

Notice that this is neat... the stop action takes place only at the end of a line. No broken lines to mess things up. At the start of a new line, we'll print a 'control-c' type of echo - up-arrow, C, and Newline - to signal the stop. Then we'll stop.

```
move.l a5,d1              ;file handle
lea    CtrlCMess(pc),a0   ;message to
move.l a0,d2              ;  buffer
moveq  #3,d3              ;length
jsr    _LVOWrite(A6)
```

It seems odd, but putting a zero into register D5, our buffer size, will stop the program. We'll connect this up in a moment.

```
         moveq #0,d5       ;zap buffer causing exit
NotCtrlC moveq #0,d7       ;Column count = 0
         bra.s NextChar
```

We come to routine EndBuf when we have used up the buffer and want to fill it up again so that we can continue with the file. How do we know when to stop? There was a hint in our discussion of *Read*.

If we are not at the end of a file, we get a full buffer. Only at the end of the file is the buffer partly full. Now: we logged the number of characters we got into D5. If it's equal to the buffer size, we may go back and do more. If not, we have finished the file. Print a NewLine to make things neat and quit.

Mental exercise: what happens if the last block exactly fills the buffer?

```
; The buffer is used up. If twas full, refill it.
EndBuf     cmp.w  #BufSize,d5   ;was it a full buffer?
           beq    ReadBlock     ; yes, get more
           move.b #$0a,(a3)     ; no, send NewLine
           move.l a5,d1         ; file handle
           move.l a3,d2         ; buffer
           moveq  #1,d3         ; length
           jsr    _LVOWrite(A6)
           rts                  ; job complete

dosname    dc.b 'dos.library',0
CtrlCMess dc.b '^C',$0a
```

Our fixed data is just above: the DOS library name and the CTRL-C echo message. Coming up next: our two buffers. The output buffer is only one character long, and the input buffer ten characters. DC.B means 'put a zero here'. DS.B means, 'set aside ten characters, but I don't care what's there'.

```
OutBuf     dc.b  0
BufSize    EQU  10
BufAdr     ds.b  BufSize
```

The assembler directive EQU ('equate') generates no code or memory. It just says, 'any time you see BufSize, take it to mean value 10'.

That's it. Assemble and link it, and you'll find you have a handy little file snooper.

Impure thoughts

The above code is not pure. If the program were trying to do two (or more) things at once, the two buffers would probably get the data mixed.

This would not happen if we just opened two separate CLI windows and commanded DUMP <filename> in each. A separate copy of program DUMP would be loaded for each job, and there would be no conflict. The conflict will arise if we make DUMP *resident*.

Assuming you have Workbench 1.3 or later, try commanding RESIDENT DUMP PURE. (PURE will override the fact that we have not set the 'pure' bit in the program file).

Now, open two Shell windows... only Shell windows use Resident commands. In each window, type DUMP <filename>. The file names may be the same or different; I would suggest text files in RAM, such as your .ASM source files. They are easy to read, and you can spot any problems quickly.

One copy of DUMP in memory tries to run both files, and, as you will see, fails miserably. There's no danger to the system in this case, but you will see that the output is a mess.

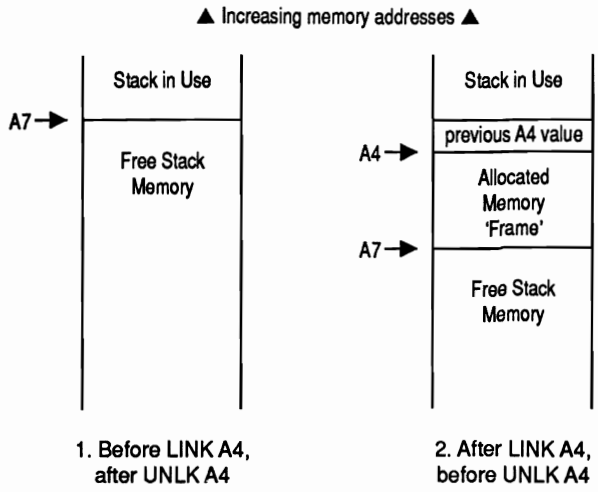


Figure 1. LINK (used with any address register) causes a 'frame' of memory to be allocated by the stack. UNLK returns the memory to the stack and restores the former value in the address register. A7 is the stack pointer, sometimes called 'SP'.

It's only the buffers that cause this effect. Is there an easy fix? Let's get our buffers from the stack.

LINK and UNLK

If the program commands LINK A4,#-$10, 16 bytes (hex 10) will be taken from the stack and reserved for your use. Register A4 will point at the *top* of this memory area; we can reach the memory by using an offset from A4.

We must be absolutely sure to return the borrowed memory when we are finished with it, using UNLK A4. Because of the way the stack works, we must return the memory in the same subroutine that asked for it.

A detailed explanation of how the register and stack work together with commands LINK and UNLK would take up too much space here. But it's a handy and dependable way of getting a modest amount of memory. See Figure 1 for an illustration of what happens.

A pure program

This is similar to the earlier version of DUMP. Only the new code will be commented.

Try this one as a resident program. You'll find it to be pure as the driven snow.

```
;
;   Dumper - file dump - Jim Butterfield.  Feburary 15/89.

; Exec library calls
            xref _LVOOpenLibrary        ;-$228
            xref _LVOCloseLibrary       ;-$19E
            xref _LVOSetSignal          ;-$132
; DOS library calls
            xref _LVOOpen               ;-$1E
            xref _LVOClose              ;-$24
            xref _LVOOutput             ;-$3C
            xref _LVOWrite              ;-$30
            xref _LVORead               ;-$2A
;
BufSize EQU 10
;-- Initial setup:
Startup     move.l a0,a4        ; Remember ptr to argument line. cf MOVEA
            move.l d0,d4                ; Remember len of argument line.
            lea    dosname(pc),a1       ;-Specify name 'dos.library'.
            clr.l  d0                   ;   Any version (0)
            move.l $4,a6                ;   Then using Exec library,  cf A
            jsr    _LVOOpenLibrary(a6)  ;   Open Dos library.
            move.l d0,a6                ; Remember dosbase ptr.
            beq.s  StartupQuit          ; Check for error (d0=0 means
            bsr.s  DOSinit
            move.l a6,a1                ;-Specify Dos library in a1;
            move.l $4,a6                ;   then using Exec library,
            jsr    _LVOCloseLibrary(a6)     ;   close Dos library.
StartupQuit rts                         ; End of Program
;
;-- Get CLI outhandle:
DOSinit     jsr    _LVOOutput(a6)       ;   get CLI outhandle,
            move.l d0,a5                ;   & then remember it.
;
; -- Change arg file name to C string:
checklen    move.b #0,-$1(a4,d4.W)      ; binary zero at end
skipspc     cmp.b  #$20,(a4)+
            beq.s  skipspc
            subq.l #1,a4
            move.l a4,d1                ; filename pointer
            move.l #1005,d2             ; MODE_OLDFILE (for reading)
            jsr    _LVOOpen(A6)
            move.l d0,d6                ; file inhandle
            beq.s  DOSquit              ; no good, quit
            link   a4,#-$10
            bsr.s  DumpFile             ; the main job
            unlk   a4
            move.l d6,d1                ; use the handle..
            jsr    _LVOClose(A6)        ; to close the file
DOSquit     rts                         ;   exit program.
;
; DOS is open and we have our in/out handles.  Let's go!
DumpFile    moveq  #0,d7                ; zero column count
ReadBlock   move.l d6,d1                ; input file handle
            lea    -$10(a4),a2          ; input buffer (stack) address
            lea    -$4(a4),a3           ; output buffer (stack) adds
            move.l a2,d2                ; copy to D2 for read
            moveq  #BufSize,d3          ; input buffer size
            jsr    _LVORead(A6)         ; so read it
            move.l d0,d5                ; size of data
; Here's where we scan and output
            moveq  #0,d4                ; start of buffer pointer
; Note that A2 still contains input buffer address
; Character loop; are we at end of buffer?
NextChar    cmp.l  d5,d4
            bcc.s EndBuf
; We have data!  Let's crunch it.
            move.b $0(a2,d4),d0
            addq.l #1,d4
            cmp.b  #$0a,d0              ;cf CMPI
            beq.s  NuLiner
            cmp.b  #$1F,d0              ;cf CMPI
            bhi.s  NotDot
            move.b #$2e,d0
NotDot      move.b d0,(a3)
            move.l a5,d1                ;file handle
            move.l a3,d2                ;outbuffer
            moveq  #1,d3                ;length
            jsr    _LVOWrite(A6)
; count and send NewLine if needed
            addq.b #1,d7                ;bump column count
```

```
            cmp.b  #64,d7            ; at limit?
            bls.s  NextChar
NuLiner     move.b #$0a,(a3)
            move.l a5,d1             ;file handle
            move.l a3,d2             ;buffer
            moveq  #1,d3             ;length
            jsr    _LVOWrite(A6)
            move.l a6,d7             ;stash DOSbase

            moveq  #0,d0
            move.l #$1000,d1
            movea.l $4,a6
            jsr    _LVOSetSignal(A6) ;test CTRL-C
            move.l d7,a6             ;restore DOSBase
            andi.l #$1000,d0
            tst.l  d0
            beq.s  NotCtrlC
            move.l a5,d1             ;file handle
            lea    CtrlCMess(pc),a0  ;message to
            move.l a0,d2             ;       buffer
            moveq  #3,d3             ;length
            jsr    _LVOWrite(A6)
            moveq  #0,d5             ;zap buffer causing exit
NotCtrlC    moveq  #0,d7             ;colcount = 0
            bra.s  NextChar
; The buffer is bare.  If twas full, refill it.
EndBuf      cmp.w  #BufSize,d5       ; was it a full buffer?
            beq    ReadBlock         ; yes, get more
            move.b #$0a,(a3)         ; no, send NewLine
            move.l a5,d1             ; file handle
            move.l a3,d2             ; buffer
            moveq  #1,d3             ; length
            jsr    _LVOWrite(A6)
            rts                      ; job complete
dosname     dc.b 'dos.library',0
CtrlCMess   dc.b '^C',$0a

   end
```


Schedule Assistant

Voice and Sound Reminders —
Create your own or use NAG Plus library.

Perpetual Calendar —
Enter 99 events per day.

Auto Dialer —
Connects YOU, not your modem.

AREXX Port —
Commands any timed event or action.

Notepad —
Click on any word to open text editor.

Print —
One click prints any event or appointment list.

AND MUCH MORE!

Suggested Retail $79.95. Ask your dealer or contact:

NAG® PLUS 3.0

Gramma software For Your Own Good
17730 15th Avenue N.E.
Suite 223
Seattle, Washington 98155
(206) 363-6417

Now For The Amiga!

Are you tired of fumbling under or behind your computer to swap your mouse and joystick cables? Are your cable and computer connectors worn out from all the plugging and unplugging? Then Mouse Master is a must for you!

$39.95
plus shipping & handling.

(602) 322-6100

Practical Solutions®
1135 N. Jones Blvd., Tucson, AZ 85716

MOUSE MASTER

NEW Release
ver 1.10

- Full Intuition based Interface
- Selective Restore
- Amiga wildcards
- Multiple indexing for greater safety
- Over 907K per floppy

saf-T-net

Hard Disk Backup
for your Amiga
available at your dealers **now**
or from
(714) 955-2555
Metadigm, Inc. 19762 MacArthur Blvd. Suite 300; Irvine CA 92715

Creative Computers

ORDERS ONLY: 800-872-8882 OUTSIDE CALIF. 213-370-2009 INSIDE CALIF. 213-542-2292 CUSTOMER SERVICE OR ORDER STATUS

VISA, MasterCard, American Express, Discover, International Phone and Mail Orders, Educational, Corporate and Aerospace Purchase Orders Accepted.
4453 Redondo Beach Blvd., Lawndale, CA 90260 Mon.-Sat. 8 a.m.-6 p.m. PST FAX: 213-214-0932

HOT NEW PRODUCTS

VIDEO & AUDIO:
- Forms in Flight II
- Photon Paint II
- Music X
- Caligari
- The Performer
- Movie Setter
- MidiVU
- C-Zar
- Comic Fonts
- Pro-Sound Designer
- MIDI Magic
- Zoetrope
- Photon Cell Animator

GAMES:
- Double Dragon
- Designasaurus
- Bool
- Trump Castle
- Operation Wolf
- Bubble Bobble
- Roston
- Dragon's Lair
- TV Sports Football
- Lords of the Rising Sun
- Univerol Military Simulator
- Advanced Dungeons & Dragons
- Kingdoms of England
- Hostage
- Test Drive 2
- Downhill Challenge
- Rampage
- Evil Garden
- Artic Adventurel

BUSINESS PRODUCTIVITY:
- Page Stream
- Works Platinum
- Pen Pal
- Tax Break
- Desktop Budget
- Pro-Net
- Pro-Board
- Professional DataRetrieve
- Professional Draw
- Aegis Draw 2000
- Superbase Personal 2
- Superplan

EDUCATIONAL:
- My Paint

UTILITIES:
- B.A.D. (hot floppy and hard disk accelerator)

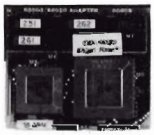
HARDWARE:
- INTRUDER ALERT
- AMAC Emulator (Macintash emulator!)
- Spirit Deluxe MIDI
- Vault-30

PLEASE CALL FOR PRICES

SOFTWARE

3-D GRAPHICS	36.13
3-DEMON	71.95
4 X 4 OFF ROAD RACING	32.46
4TH & INCHES CONSTRUCTION	9.95
4TH AND INCHES	29.24
64 EMULATOR 2, THE	49.95
A B ZOO	25.99
A-TALK III	64.97
AAARGH!	23.95
AC BASIC COMPILER	134.06
AC FORTRAN	199.00
ACCOUNTANT, THE	186.95
ADRUM	51.98
ADVENTURES OF SINBAD	32.46
AEGIS ANIMATOR	87.95
AEGIS DRAW 1.4	52.95
• AEGIS DRAW 2000	181.95
AESOP'S FABLES	19.95
ALGEBRA I	36.95
ALGEBRA II	38.95
ALIEN FIRES	24.95
ALIEN SYNDROME	32.46
ALL ABOUT AMERICA	38.95
ALTERNATE REALITY	27.95
• AUDIOMASTER II	64.95
• AUTODUEL	27.95
AWARD MAKER PLUS	31.16
AWARD MAKER: SPORTS	19.95
AZTEC 68/AM-D	194.95
AZTEC C PROFESSIONAL	129.95
• AZTEC 68SK CBBK SPEC.	369.72
• B.A.D. DISK OPTIMIZER	32.46
• BAAL	27.95
BAD CAT	30.95
BALANCE OF POWER	34.95
BALANCE OF POWER (BOOK)	8.51
BARBARIAN	25.95
BARD'S TALE	35.95
BARD'S TALE CLUE BOOK	10.95
BARD'S TALE II	41.95
BARD'S TALE II CLUE BOOK	10.50
BARDS TALE HINT DISK	13.95
BASIC GRAMMER SERIES	19.46
• BATTLE CHESS	32.46
BATTLE SHIP	19.46
BBS-PC	62.32
BECKER TEXT	99.95
BENCHMARX C LIBRARY	64.95
BENCHMARX IFF LIBRARY	64.95
BENCHMARX MODULA-2	129.95
BENCHMARK SIMPLE LIBRARY	64.95

MIDGET RACER CSA



HOT NEW ACCELERATOR FOR ALL AMIGAS $349.00
- 68020 processor.
- Optional math co-processor up to 33 Mhz 68882!
- 30% average speed increase *without* math co-processor.

ALSO: Sculpt Animate 4D $399.95

PACKAGE DEAL $979.00
INCLUDES: 1. SCULPT ANIMATE 4D; 2. MIDGET RACER; 3. 68882-20 CO-PROCESSOR; 4. 20 OR 24 Mhz CRYSTAL OSSC.

AMIGA DOS EXPRESS	19.95
AMIGA FONTS (NTF)	59.95
AMIGA KARATE	24.95
ANALYTIC ART-GRAPHICS	37.48
ANALYZE 2.0-SPREADSHEET	93.73
ANDROMEDA MISSION	25.95
ANIMAL KINGDOM	32.46
ANIMATE-3D	99.95
ANIMATION	62.95
ANIMATION EDITOR	38.95
ANIMATION EFFECTS	32.46
ANIMATION MULTIPLANE	58.46
ANIMATION STAND	32.46
ANIMATOR'S APPRENTICE	184.95
ANNALS OF ROME	24.95
ANTI-VIRUS	25.95
ARAZOK'S TOMB	31.25
ARCADE ACTION PACK	34.95
• ARENA	9.95
• ARKXX	32.95
ARKANOID	19.95
ART COMPANION	19.95
ART GALLERY FANTASY	23.36
ART OF CHESS	22.95
ASHA'S FONTS	58.95
ASSEMPRO	71.95
ATREDES (BBS)	97.46
BETTER DEAD THAN ALIEN	21.95
BEYOND ZORK	33.76
BIONIC COMMANDO	25.97
BLACK CAULDRON	28.95
BLACK JACK ACADEMY	29.95
BLACK LAMP	22.00
BLACK SHADOW	22.95
BLITZKRIEG (1 MEG VER.)	38.95
BLOCKBUSTER	32.46
BOBO	23.36
BOOMERAID	32.46
BREACH	25.95
BREACH SCENARIO DISK	16.95
BRIDGE 5.0	24.10
BRUSH WORKS	20.95
BRUSH WORKS 2	19.95
BUBBLE GHOST	22.71
BUMPER STICKER MAKER	37.95
BUTCHER 2.0	23.13
CALIGHT	38.95
• C-ZAR	126.95
CAD PARTS FOR INTRO CAD	16.50
CALCULUS	36.95
• CALIFORNIA GAMES	32.46
• CALIGARI	CALL
CALLIGRAPHER	84.95
CAPE 68K ASSEMBLER 2.0	58.47
CAPITALIZATION SERIES	19.46
CAPONE	25.95
CAPTAIN BLOOD	32.46
CAPTAIN FIZZ	19.95
CARRIER COMMAND	30.95
CASINO FEVER	25.95
CB TREE PLUS	64.95
• CBM DOS 1.3 WORKBENCH	24.95
CELEBRITY COOKBOOK	22.95
CENTERFOLD SQUARES	19.95
• CHARON 5	25.96
CHESSMASTER 2000	32.46
CHICKEN LITTLE	19.95
CHRONO QUEST	33.95
CITY DESK	93.95
CITY DESK ART COMPANION	19.95
CITY DESK BUNDLE	99.95
CLI MATE	24.95
CLIP ART #1-#8 (EA.)	12.95
COMBAT COURSE	25.95
• COMIC FONTS (NEW & NICE)	59.95
• COMIC SETTER	64.96
COMIC SETTER (FUNNY DATA)	22.95
COMIC SETTER (SF DATA)	22.95
COMIC SETTER (SUPERHEROS)	22.95
• COSMIC BOUNCER	19.95
CRAPS ACADEMY	25.95
CRASH GARRET	25.96
CRAZY CARS	25.95
CROSSWORD CREATOR	34.95
• CUBEMASTER	9.95
CUSTOMS SCREENS	43.73
• CYGNUS ED PROF.	64.95
DARK CASTLE	25.95
DATA RETRIEVE	49.95
• DATA RETRIEVE PROF.	211.95
DEATH SWORD	19.95
DECIMAL DUNGEON	32.46
• DEEP SPACE	9.95
DEFCON 5	25.95
DEFENDER OF THE CROWN	34.95
DEJA VU	34.95
DELUXE ART PARTS #1	21.95
DELUXE ART PARTS #2	21.95
DELUXE HELP CALLIGRAPHER	22.71
DELUXE HELP FOR DIGIPAINT	21.84
DELUXE HELP FOR DPAINT II	21.84
DELUXE HELP FOR PHOTON	21.95
DELUXE HELP PAGE SETTER	22.95
DELUXE MAPS	16.95
DELUXE MUSIC 2.0	69.95
DELUXE PAINT ART & UTIL	21.95
DELUXE PAINT II	89.95
DELUXE PHOTO LAB	99.95
DELUXE PRODUCTIONS	139.95
DELUXE VIDEO 1.2	89.95
DEMONSTRATOR	21.95
DES CARTES	22.95
DESIGN DISK:	
ARCH SCULPT or VSCAPE	22.74
FUTURE SCULPT or VSCAPE	22.74
HUMAN SCULPT or VSCAPE	22.74
MECHO SCULPT or VSCAPE	22.74
• DESIGNASAURUS	32.46
DESKTOP ARTIST	18.95
• DESKTOP BUDGET	45.95
DESTROYER	25.95
DEV DISK (ASSORTED UTIL)	7.95
DEVELOPERS TOOLKIT	36.95
DEVPAC AMIGA ASSEMBLER	64.95
DIGI-PAINT	41.95
DIGI-VIEW UPGRADE	13.95
• DIRECTOR'S TOOLKIT	24.95
DIRECTOR, THE	45.95
DISCOVERY EXPANSION DISKS	12.97
DISCOVERY MATH	25.00
DISCOVERY SPELL	25.00
DISCOVERY TRIVIA	25.00
DISCRETE MATHEMATICS	36.95
DISK MECHANIC	58.50
DISK PRO PLUS	18.95
DISK TO DISK	34.95
DISK WICK	32.46
DISKMASTER	37.95
DISTANT ARMIES	29.95
DOCTOR TERM PROFESSIONAL	74.95
DOMINOES	16.95
DOS TO DOS	37.95
• DOUBLE DRAGON	27.95
DOUGS MATH AQUARIUM	31.95
DOUGS COLOR COMMANDER	19.46
DR. TS:	
4-OP ED/LIB	92.95
BACH SONGBOOK	17.95
CASIO V21 ED/LIB	92.95
COPYIST	155.38
CZ CARTRIDGE	51.95
CZ RIDER	92.95
D-110 ED/LIB	92.95
D-50 ED/LIB	92.95
DR. DRUMS	17.95
DR. KEYS	17.95
DX7 ED/LIB	92.95
ESO-APADE ESO-1	92.95
KAWAI K-1 ED/LIB	92.97
KAWAI K-5 ED/LIB	92.95
KCS	154.95
LEXICON ED/LIB	92.95
MIDI RECORD. STU.	43.50
MT-32 ED/LIB	92.95
MATRIX-6 ED/LIB	92.95
• DRAGON'S LAIR	46.50
DREAM ZONE	32.46
DRUM STUDIO	32.47
• DUNGEON MASTER	32.46
DUNGEON MASTER HINT DSK.	12.95
DYNAMIC DRUMS	49.95
DYNAMIC STUDIO	139.00
DYNAMIC CAD	299.00
EARL WEAVER BASEBALL	34.95
EARL WEAVER STATS	12.95
EASY LOANS	24.95
EBON STAR	25.96
EMERALD MINES	14.95
EMETIC SKIMMER	25.95
EMPIRE	34.32
ENCORE	45.95
EUROPEAN SCENERY DISK	27.95
• EVIL GARDEN	25.95
• EXCELLENCE	159.00
EXPLORER	32.47
EXPRESS COPY	29.95
EXPRESS PAINT 3.0	90.95
EXTEND	25.95
FACC II	21.95
FAERY TALE ADVENTURE	31.95
FAERY TALE GUIDEBOOK	7.95
FALCON	32.46
• FAMILY TREE	32.46
FANCY 3D FONTS	46.95
FANTAVISION	47.95
FERRARI FORMULA ONE	33.95
• FINAL ASSAULT	32.46
FINAL MISSION	19.95
FINE PRINT	41.95
FIRE 'N FORGET	25.95
FIRE ZONE	23.95
FIRE BRIGRADE	32.46
FIREPOWER	16.95
FIRST LETTERS & WORDS	33.00
FIRST SHAPES	33.00
FISH	29.95
FLEET CHECK	25.96
FLIGHT PATH 737	16.95
FLIGHT SIMULATOR II	37.46
FLOW	62.32
FONTS AND BORDERS	22.95
FORTHAM	29.95
• FORMS IN FLIGHT II	79.50
FOUNDATION'S WASTE	23.62
FRACTION ACTION	32.46
GALACTIC CONQUERER	29.95
• GALAXY FIGHT	9.95
GALILEO 2.0	45.46
GALLERY 3-D (SCULPT)	51.95
GAMES, THE WINTER EDITION	32.46
GETTYSBURG	38.95
• GFA BASIC 3.0	90.95
GIZMOZ	45.95
GLOBAL COMMANDER	27.46
GOLD SPELL II	29.95
GOLDEN PATH	29.95
GOLDRUNNER II	25.95
GOMF! 3.0 (SOFTWARE ONLY)	27.95
GRABBIT	20.95
GRADE MANAGER	58.95
GRAND SLAM TENNIS	31.25
GRAPHICS STUDIO	38.95
GREAT STATES II	25.95
• GRETZKY HOCKEY	32.46
GUILD OF THIEVES	30.95
HAMANOEL	25.95
HAROBALL	28.95
HARRIER COMBAT SIMULATOR	32.95
HEAD COACH	32.46
HEROES OF LANCE HINT DSK.	13.95
• HEROES OF THE LANCE	27.95
HISOFT BASIC PROF.	103.95
HIT DISK VOL 1	32.46
HOLE IN ONE	25.99
HOLLYWOOD POKER	25.95
HOME BUILDERS CAD	129.95
HOSTAGE	29.95
HOT & COOL JAZZ	21.95
HUNT FOR RED OCTOBER	34.95
• HYBRIS	25.99
ICON PAINT	58.95
IMPACT-BUSINESS GRAPHICS	62.46
IMPOSSIBLE MISSION II	32.46
INDOOR SPORTS	31.22
INOVA TOOLS	52.03
INSTANT MUSIC 1.2	33.00
INTELLITYPE	35.95
INTERCEPTOR	37.95
INTERCHANGE	29.95
INTERCHANGE CONVERSION	16.95
INTERCHANGE OBJECTS #1	18.95
INTERCHANGE TSLV MODULE	12.96
INTERFONT	75.38
INTERNATIONAL SOCCER	25.96
INTROCAD	49.95
INVESTOR'S ADVANTAGE	64.95
• INVISION	99.95
IT'S ONLY ROCK & ROLL	21.95
J FORTH	77.96
JET	37.46
JET SET FONT SET	32.50
JINXTER	25.95
• JOKER POKER	32.46
KAMPFGRUPPE	41.95
KARA FONTS	54.95
KARATE KID II	25.95
KEY TO 'C'	22.95
KICK-WORK 1.3	27.95
KIDTALK	27.95
KIKUGI	18.95
KINDERAMA	32.46
KINDWORDS	64.95
KING OF CHICAGO	34.95
MICROLAWYER	37.95
• MIDIVU	29.95
• MISSION CON-BAT	31.16
MODELER 3-D	65.00
MOEBIUS	39.95
MONEY MENTOR	64.95
• MOVIE SETTER	64.95
MULTI-FORTH	59.95
MULTI-PREFS	19.45
MURAL PRINTER	29.95
MUSIC MOUSE	51.95
MUSIC STUDENT	37.95
MUSIC STUDIO	34.95
• MY PAINT	32.46
NAG PLUS 3.0	49.95
NEWSLETTER FONTS COLOR	29.95
OBLITERATOR	25.95
OFFSHORE WARRIOR	25.96
OGRE	32.46
ONLINE 2.0	43.56
OPERATION CLEAN STREETS	25.95

We carry OVER 1100 items
The best NEW products
The LOWEST PRICES

NEW PRODUCTS ARRIVE DAILY.
CALL FOR UNLISTED ITEMS.

SPECIAL HOT PRODUCT!
DELUXE PAINT III $99.95

KING'S QUEST I	32.95
KING'S QUEST II	32.95
KING'S QUEST III	32.95
KNIGHT ORC	30.95
LANCELOT	27.95
LASER UP! FONTS VOL. I	31.95
LASER UP! PLOT 1.2	39.95
LASER UP! PRINT 1.2	71.95
LASER UP! UTIL. VOL. I	31.95
• LATTICE "C" 5.0 COMPLETE!	195.00
• LATTICE C++	345.00
LAZERSCRIPT	28.16
LEADER BOARD DUAL PACK	22.72
LEADER BOARD TORNA DISK	13.95
LEADER BOARD-GOLF GAME	27.00
LEARNING CURVE	51.95
LEARNING THE ALPHABET	19.46
LEATHER GODDESSES	32.46
LEATHERNECK	25.95
LEISURE SUIT LARRY	26.40
LEXCHECK	26.85
LIFE CYCLES AMIGA	19.46
LIGHTS! CAMERA! ACTION!	59.95
LINKWORD DUTCH	20.55
LINKWORD FRENCH	20.55
LINKWORD GERMAN	20.55

Forms In Flight II ($79.50)

LINKWORD GREEK	20.55
LINKWORD ITALIAN	20.55
LINKWORD PORTUGUESE	20.55
LINKWORD RUSSIAN	20.55
LINKWORD SPANISH	20.55
LINT (GIMPLE)	63.70
LION'S/CALLIGRA FONTS	58.95
LITTLE RED HEN	19.95
LOGIC WORKS ADVANCED	189.95
LOTTERY MAGIC	19.95
LURKING HORROR	25.95
• M INTELLIGENT MUSIC	143.46
• MAGELLAN	119.00
MAGICAL MYTHS	32.46
MAJOR MOTION	25.96
MANIAX	16.95
MARAUDER II-DUPLICATION	27.47
MARBLE MADNESS	33.00
MASTER NINJA	30.95
MASTERING CLI	25.95
MASTERPIECE FONT 20-DSKS	167.95
MATCH IT	25.95
MATH MAGICIAN	27.49
MATH TALK	31.95
MATH TALK FRACTIONS	26.95
MATH WIZARD	32.46
MATH-AMATION	64.95
MAVIS BEACON TYPING	30.89
MAXIPLAN 500	93.95
MAXIPLAN PLUS	124.95
MEAN 18 COURSE DISK	14.96
MEAN 18 COURSE DISK II	14.95
MEAN 18 GOLF	28.95
MEDIALINE FONTS	22.95
MENACE	19.98
METACOMCO ASSEMBLER	68.85
METACOMCO PASCAL VER 2.0	92.95
METACOMCO SHELL	39.95
METACOMCO TOOLKIT	34.95
METASCOPE DEBUGGER	59.95
MICROFICHE FILER	69.95
MICROFICHE FILER PLUS	118.95
ORGANIZE	62.32
OTG DSM (DISASSEMBLER)	43.95
OUT RUN	32.46
P.D.W.	25.95
PAGE FLIPPER	31.23
PAGE FLIPPER PLUS F/X	103.95
• PAGE STREAM	129.95
PAGESETTER	93.72
PALADIN	25.95
PALADIN (QUEST DISK 1)	16.86
PCLO PRINTD CURCUIT LYOUT	199.95
PCLO+ PRNTD CURCUIT LYOUT	399.95
PERFECT SCORE	54.95
• PERFORMER (ELAN)	38.95
PHANTASIE	27.47
PHANTASIE 3	24.95
PHANTASM	22.95
• PHASAR-FIN'L MGMT	62.48
• PHOTON CELL ANIMATOR	99.95
PHOTON PAINT	64.96
PHOTON PAINT EXP. DSK.	19.95
PHOTON PAINT TRANS. CNTRL	194.95
PHOTOSYNTHESIS	97.95
PINBALL I Q	19.46
PIONEER PLAGUE	25.96
PIXMATE	45.95
PLANET PROBE	19.95
PLATOON	29.21
POKER SOLITAIRE	22.95
POLICE QUEST	32.46
PORTS OF CALL	29.95
POWER WINDOWS 2.5	58.47
POWERSTYX	22.95
PRE CALCULUS	36.95
PRESIDENT IS MISSING	25.95
PRIME TIME	25.95
PRINTMASTER PLUS	31.23
PRO MIDI STUDIO	130.95
• PRO VIDEO PLUS	184.95
PROBABILITY THEORY	36.95
• PROFESSIONAL DRAW	129.95
• PROFESSIONAL PAGE	229.00
PROJECT D	31.95
PROMISE SPELLING CHECKER	34.95
PROSCRIPT	32.95
PROWRITE 2.0	78.10
PUNCTUATION SERIES	19.46
QUADRALIEN	19.95
QUARTERBACK	45.95
QUESTRON II	32.46
QUICK MERGE	37.95
QUINTETTES	29.95
QUIZ MASTER	49.95

Boing! Mouse

Professional Optical 3-button mouse for your Amiga.
- Optical technology eliminates friction and momentum coefficients for super-precise handling.
- No moving parts for increased reliability.
- Middle button makes it the only mouse compatible with A2024 software and X Windows.
- Compatible with ALL Amiga computers.

SPECIAL PRICE!
LIMITED TIME ONLY: $99.95

WE CAN BEAT ANY ADVERTISED PRICE! BUT WE SELDOM HAVE TO — OUR PRICES ARE THE LOWEST! (IF YOU SEE A LOWER PRICE ON AN ITEM, GIVE US A CALL)

R.R. AESOP'S FABLES ... 19.95
RAW COPY 1.3 ... 38.95
REACH FOR THE STARS ... 27.95
READ & RHYME ... 31.95
READ-A-RAMA ... 32.46
REALM OF THE WARLOCK ... 22.74
REBEL CHARGE CHICKAMANGA ... 39.95
REEL FISHIN'—GONE FISHIN ... 30.88
RETURN TO ATLANTIS ... 34.95
ROAD RAIDERS ... 25.96
ROADWAR 2000 ... 27.47
ROADWAR EUROPA ... 29.95
ROADWARS ... 23.95
ROB READER UGLY DUCKLING ... 19.45
ROBBEARY ... 16.21
• ROCK CHALLENGE ... 25.95
ROCKET RANGER ... 32.46
ROCKFORD ... 23.95
ROMANTIC ENCOUNTER ... 25.95
SAMPLEWARE-DIGITAL ... 16.95
SAMPLEWARE-GRAB BAG ... 16.95
SAMPLEWARE-ORCHESTRAL ... 16.95
SAMPLEWARE-ROCK ... 16.95
SANTA PARAVIA & FIUMACCIO ... 19.95
SARGON III ... 35.95

SCARY MUTANT SPACE ALIENS ... 25.95
SCENERY DISK #11 EAST COAST ... 18.72
SCENERY DISK #7 EAST COAST ... 18.71
SCHEMATIC CAPTURE ... 199.95
SCRABBLE ... 27.46
SCRIBBLE ... 62.32
SCULPT 3-D ... 69.95
• SCULPT ANIMATE 4-D ... 399.95
• SDL CINAMAWARE SERIES ... 34.95
SEASONS AND HOLIDAYS ... 21.95
SECONDS OUT ... 19.46
SECURITIES ANALYST ... 49.95
SEVEN CITIES OF GOLD ... 14.95
SEX VIXENS ... 25.95
SHADOW GATE ... 31.95
SHAKESPEARE ... 146.95
SHANGHAI-STRATEGY GAME ... 27.46
SHERLOCK ... 27.00
SILENT SERVICE ... 25.95
SINBAD & FALCON ... 34.95
SKY BLASTER ... 19.95
SKY CHASE ... 26.95
SKYFOX II ... 27.45
• SLAYGON ... 14.95
SMOOTH TALKER ... 33.95
SNAKE PIT ... 24.95
SOCIAL STUDIES ... 19.46
SOFTWOOD FILE SG ... 78.21
SOLITAIRE ROYAL ... 22.71
SONIX (AEGIS) ... 49.95
SORCERER LORD ... 23.95
SOUND OASIS ... 64.95
SOUNDLAB ... 209.95
SOUNDQUEST:
CZ MASTER ... 92.95
0-10 MASTER ... 92.95
0-110 MASTER ... 92.95
0-50 MASTER ... 92.95
DX7 MASTER ... 109.95
DX7-II MASTER ... 109.95
ESQ/ESQ-80 MST ... 92.95
GENERIC MASTER ... 77.95
K-1 MASTER ... 92.95
M-1 MASTER ... 92.95
MT-32 MASTER ... 92.95
TEXTURE ... 92.95
TX-802 MASTER ... 92.95
TX-81Z MASTER ... 92.95
SOUNDSCAPE PATTERN SPLAT ... 35.95
SOUNDSCAPE UTILITIES 1 ... 35.95
SOUNDSCAPE UTILITIES II ... 42.95
SOURCE LEVEL DEBUGGER ... 49.96
SPACE CUTTER ... 19.95
SPACE HARRIER ... 32.46
SPACE MATH ... 29.95
SPACE PORT ... 27.38
SPACE QUEST II ... 32.46
SPACE RACER ... 19.46
SPACE SCHOOL SIMULATOR ... 22.95
SPACE STATION OBLIVION ... 32.46
SPACEQUEST ... 33.00
SPELLBOUND ... 25.95
SPELLER BEE ... 31.25
SPELLING IMPROVEMENT ... 25.95
STAR WARS EMPIRE STRIKES ... 30.25
STARFLEET I ... 29.95
STARGLIDER ... 30.95
• STARGLIDER II ... 29.95
STARRAY ... 19.95
STELLAR CONFLICT ... 26.95
STELLARYX ... 25.95
STOCK MARKET-THE GAME ... 16.95
STRIP POKER ... 27.46
STRIP POKER DATA #4 ... 12.95
STRIP POKER DATA #5 ... 12.95
STRIP POKER II ... 25.95
STUDIO FONTS V1 (COLOR) ... 29.95
STUDIO MAGIC ... 64.95
SUB BATTLE ... 24.95
SUMMER CHALLENGE ... 19.95
SUPERBACK ... 51.95

SUPERBASE ... 93.73
• SUPERBASE PERS. II ... 94.46
• SUPERBASE PROF. 3 ... 209.95
• SUPERPLAN (SPREADSHEET) ... 99.95
SUPERSTAR ICE HOCKEY ... 32.46
SWORDS OF SODAN ... 32.48
SYMPHONY SONGS (EACH VOL) ... 15.95
SYNTHIA ... 59.95
SYSTEMS MONITOR ... 28.95
T SHELL ... 32.46
TAKE STOCK ... 32.46
TALES FROM ARABIAN NIGHTS ... 32.46
TALKING ANIMATOR ... 39.00
TALKING COLORING BOOK ... 18.95
TANGLE WOOD ... 25.95
TARGIS ... 28.95
TATE FONTS ... 45.95
• TAX BREAK ... 51.95
TECHNOCOP ... 32.46
• TELE EPIC ... 25.96
• TELE WARS II ... 25.96
TELEGAMES ... 23.95
TERA QUESTRON ... 25.95
TERRAIN FOR TURBO SILVER ... 19.95
TERRORPODS ... 25.95

TEST DRIVE ... 34.95
TETRA QUEST ... 25.95
TETRIS ... 21.80
TEXTCRAFT PLUS ... 59.95
THEXDER ... 23.95
THINKER ... 38.95
THREE LITTLE PIGS ... 19.95
THREE STOOGES ... 35.95

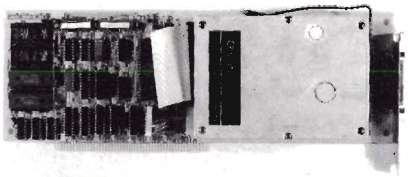
THUNDERBOY ... 22.95
TIME & MAGIK ... 26.95
TIME BANDITS ... 21.95
TOOL CADDY ... 32.46
• TOWER TOPPLER ... 32.46
TRACERS ... 22.95
TRANSCONTINENTAL RAILROAD ... 25.95
TRANSPUTOR ... 22.95
TRIANGO ... 29.95
TRIGONOMETRY ... 36.95
TRINITY ... 25.95
TRUE BASIC ... 68.95
TRUE BASIC ARITHMETIC ... 45.95
TRUE BASIC TRUE STAT ... 32.46
• TRUMP CASTLE ... 25.95
TURBO PRINT ... 32.95
• TURBO SILVER 3.0 ... 129.95
TV SHOW ... 64.95
• TV SPORTS FOOTBALL ... 34.95
TV TEXT ... 62.32
TWILIGHTS RANSOM ... 19.95
TXEO PLUS ... 51.95
ULTIMA III ... 28.95
ULTIMA IV ... 38.96
ULTRA DOS ... 37.95
UMS CIVIL WAR DATA DISK ... 12.95
UMS VIETNAM DATA DISK ... 12.95
UNCLE O CON SOUND TRATION ... 25.97
UNINVITED ... 34.95
• UNIVERSAL MILITARY SIMULTR ... 32.46
VAMPIRE'S EMPIRE ... 29.95
• VIDEO EFFECTS 30 ... 129.95
VIDEO VEGAS ... 24.10
VIDEO VISIONS 1 ... 12.95

VIDEO VISIONS 2 ... 12.95
• VIDEOSCAPE 3D ... 124.95
VIDEOTITLER ... 99.95
VIP VIRUS ... 32.46
VIRUS (GAME) ... 19.46
VOCABULARY IMPROVEMENT ... 25.95
W SHELL ... 31.95
WAR & MIDDLE EARTH ... 36.99
WARLOCK ... 22.71
WBEXTRAS ... 21.95
WESTERN GAMES ... 31.95
WHO FRAMMED ROGER RABBIT ... 29.95
WILLOW ... 25.95
WINDOW PRINT II ... 22.95
WINTER CHALLENGE ... 19.95
WINTER GAMES ... 27.46
WORD MASTER ... 32.46
WORD PERFECT ... 179.00
WORD PERFECT LIBRARY ... 84.95
WORLD CLASS LEADER BDARD ... 25.97
WORLD GAMES ... 27.46
WORLD TOUR GOLF ... 27.45
WRITE 'N FILE ... 59.95
• WSHELL ... 32.95
X-CAD ... 389.00
ZAK MCKRACKEN ... 29.95
ZANY GOLF ... 27.95
ZERO GRAVITY ... 19.46
ZING KEYS ... 31.95
ZING! SPELL ... 57.95
• ZOETROPE ... 89.95
ZOOM ... 21.95
ZORK TRILOGY ... 47.95
ZOUND SOUNDS VOL I ... 21.95
ZUMA FONTS VOLS 1 THRU 4 ... 68.95
ZYNAPS ... 23.95

NEW FROM CENTAUR: THE SILENT DRIVE $159

FEATURES: • CSI 3200 • Clickless • Small compact design • Metal case • Compatible with all Amiga computers • Low power consumption • Hinged dust cover • Drive disable switch • 1-year warranty.

ALSO AVAILABLE: CSI 2200 internal drive for Amiga 2000—100% compatible with same mechanism as Commodore 2010 ... $149.00



GVP GREAT VALLEY PRODUCTS

For Amiga 2000:
IMPACT HC/45—48 MB 28ms Hard Card ... 799.00
IMPACT HC/40Q—42 MB 11ms Hard Card ... 799.00
IMPACT HC/80Q—84 MB 11ms Hard Card ... 1199.00

For Amiga 500:
IMPACT A500-SCSI/HD20—20 MB HD ... 749.00
IMPACT A500-SCSI/HD45—45 MB HD ... 995.00
IMPACT A500-SCSI/HD40Q—42 MB 11ms High-Speed Hard Drive ... 1049.00
IMPACT A500-SCSI/HD80Q—84 MB 11ms High-Speed Hard Drive ... 1449.00
Quantum 80S, 84 MB 11ms 3.5" HD ... 995.00
Shock mounted, 64KB cache (for 11ms speed), SCSI interface. Compatible with IMPACT or A2090 boards.
Quantum 40S—42 MB version of above drive ... 599.00

AMAS MIDI SAMPLER ... 109.95
AMIGEN GENLOCK ... 149.00
AMPLIFIED SPEAKERS 2X50 W. ... 84.95
APRO DRAW 12X12 NO CURSOR ... 464.95
APRO DRAW 12X12 W/CURSOR ... 499.95
• BONO OPTICAL MOUSE ... 99.95
CA-880 FLOPPY DRIVE ... 199.00
CBM A2300 INT GENLCK 2000 ... CALL
CBM A520 COMPOS. ADAP 500 ... 49.95
• CBM DOS 1.3 ROM ... 39.95
CLTD TIMESAVER ... 57.95
• CMI MIDI I (EXT) ... 52.76
COLOR SPLITTER ... 79.95
CPS 500-POWER SUPPLY A500 ... 74.95
CSA DRAGSTRIP RAM ACCEL. ... 723.45
CSA FASTRACK 68020 ACCEL ... 905.45
CSI 2200 INT DRIVE A2000 ... 149.00
CSI 3200 AMIGA EXT FLOPPY ... 159.00
DELUXE MIDI INTERFACE ... 59.95
DESKJET 128K RAM CART ... 117.95
DESKJET PRINTER (300 DPI) ... 749.00
• DIGI VIEW GOLD A500/A2000 ... 149.00
DIGI-VIEW ... 143.95
EASYL 500 TABLET ... 369.00
EASYL 1000 TABLET ... 369.00
EASYL 2000 TABLET ... 369.00
ECE MIDI 500/2000 ... 48.95
ECE MIDI 1000 ... 48.95
ESCORT 48 MB HRD DR A500 ... 879.00
ESCORT 48 MB HRD DR A1000 ... 859.00
• FLICKER FIXER (HARDWARE) ... 479.00
FRAMEGRABBER ... 539.95
FUTURE SOUND A500 ... 93.95
FUTURE SOUND-AUDIOSAMPLER ... 142.20
GEN ONE GENLOCK ... 739.00
GOMF³ 3.0 W/BUTTON ... 48.95
GVP A2000 AUTOBOOT EPROM ... 29.95
GVP A500 RAM/ROM MODULE ... 69.95
HURRICANE 68030 ADAP BRD ... 300.00
HURRICANE A1000 ACCEL BRD ... 350.00
HURRICANE A1000 MEM. BRD ... 540.00
HURRICANE A2000 ACCEL BRD ... 500.00
HURRICANE A2000 MEM BRD ... 540.00
IMPRINT ... 399.00
• INTRUDER ALERT ... 124.02
LASERJET II-D PRINTER ... 2965.00
LASERJET SERIES 2 PRINTER ... 1749.00
LASERJET-2 RAM, 1 MB ... 419.95
LENS, 16 MM, ND IRIS F1.6 ... 21.85
LENS, 18 MM, W/IRIS F1.6 ... 66.25
LIGHT PEN ... 97.95
LIVE! 500 ... 269.00
LIVE! 1000 ... 270.00
• LIVE! 2000 ... 349.00
MAGNI 4004 GENLOCK ... 1595.00
MICROBT. 4 X POP-SIMM BRD ... 74.50
MICROBT 8-UP RAM BRD, 0K ... 145.00

MORE CREATIVE PROMOTIONS:

NEW NEW NEW

SCULPT-ANIMATE 4D JR. ... $99.95
CENTAUR TRUE FLIGHT ... $59.95
(Proportional control yake for Flight Simulator II)

Computer systems & package deals available ... CALL FOR DETAILS!

- **FREE VCR with purchase of computer — call for details.**
- **Amiga Live! 2000 ... $349**
- **Flicker Fixer (A2000 only) — *Eliminate all flicker and scanline gaps 100% — optional multisync monitor* ... $479**

M.A.S.T. PRODUCTS

MINIMEGS: Expandable, external FAST RAM (up to 2-M8s), low power — available for Amiga 500 or 2000 —
- **MINIMEGS 512K ... $299**
- **MINIMEGS 1 MB ... $449**
- **MINIMEGS 2 MB ... $749**

TINY TIGER: Low cost, high performance, SCSI hard drives for all Amigas — complete — nothing else to buy —
- **TINY TIGER 30 MB ... $629**
- **TINY TIGER 40 MB (Quantum) ... $799**
- **TINY TIGER 80 MB ... $1129**

UNIDRIVE ... $159
TWIN DRIVE ... $269
(special price/limited time only)

MICROBT HARDFRAME SCSI ... 239.00
MIDI GOLD ... 64.20
MIDI GOLD 2000 (INT) ... 68.47
MIMETICS MIDI INTERFACE ... 43.00
NERIKI GENLOCK ... 1975.00
OKIMATE 20 W/PLUG N PRNT ... 249.00
OKIMATE PLUG N PRINT ... 79.95
OMEGA STR. SAMPLER A1000 ... 79.95
PAINTJET COLOR PRINTER ... 1095.00
PANASONIC WV1410 CAMERA ... 225.00
PANASONIC WV1500 CAMERA ... 320.00
PEOPLE METER BIO FEEDBACK ... 39.95
PERFECT SOUND DIGITIZER ... 67.95
• PERFECT VISION ... 169.95
PRO RAM A2000 2/8 MG ... 699.00
• PRO SOUND DESIGNER ... 119.95
PROCESSOR ACCEL CMI ... 159.00
PROGEN ... 399.00
QUANTUM 40 MB DRIVE, 11MS ... 599.00
QUANTUM 80 MB DRIVE, 11MS ... 995.00
• ROLLAND MT-32 TONE MOD. ... 549.00
RUGGEDWRITTER HP PRINTER ... 1395.00
SOUND SAMPLER ... 86.95
SPIRIT 0 MB FOR A1000 ... 249.00
• SPIRIT DELUXE MIDI ... 214.95
STAR NX1000 PRINTER ... 199.00
STAR NX1000 RAINBOW ... 279.95
STARBOARD 2 SCSI MOD. ... 100.00
STARBOARD 2 UPPER DECK ... 80.00
STARBOARD 2/A500 0K ... 300.00
STARBOARD 2/A1000 0K ... 250.00
STARBOARD SB2000 ADAPTER ... 42.00
STEREO SOUND SAMPLER STRV ... 87.95
• SUPERGEN ... 699.00
• SUPRA 2400 MODEM ... 149.00
SUPRA DRIVE 20 MEG A500 ... 699.00
SUPRA DRIVE 20 MEG A1000 ... 699.00
SUPRA DRIVE 30 MEG A500 ... 799.00
SUPRA DRIVE 30 MEG A1000 ... 799.00
TOOL BOX, ESCORT FOR A500 ... 170.00
TRUMP CARD SCSI CNTR 2000 ... 168.96
• TWIN-DRIVES EXT 3.5" FDD ... 269.00
• UNIDRIVE 3.5" EXT. DRIVE ... 159.00
• VAULT-30 HARD DRIVE 30 MB ... CALL
VI 500 VIDEO INTERFACE ... 59.95
VI 500 VIDEO/RF MOD. ... 69.95
VI 2000 VIDEO INTERFACE ... 69.95
VI 2000 VIDEO/RF MOD. ... 79.95
X-SPECS 30 ... 99.95
XEROX 4020 INK JET COLOR ... 1140.00
XEROX 4020 STARTER KIT ... 156.00

ACCESSORIES

A1084 CARRING CASE ... 51.95
A500 & MONITOR CARRY BAGS ... 69.95
AMIGA BAG ... 29.95
AMIGA BRASS KEY RING ... 3.95
AMIGA FASHION GOLF SHIRT ... 19.95
AMIGA FASHION JACKET ... 39.50
AMIGA MUG ... 6.95
AMIGA NOTE PAD!! ... 4.99
AMIGA PEN & PENCIL SET ... 19.95
AMIGA PINS ... 4.95
AMIGA PLASTIC VISOR ... 1.95
AMIGA SLEEVE TSHIRT ... 7.95
AMIGA TERRY CLOTH VISOR ... 5.50
AMIGA TOWEL ... 15.95
AMIGA TWILL CAP W/CORD ... 7.95
BOSS JOYSTICK ... 12.95
CABLE, 10' CAMERA ... 12.95
CABLE, 500/2000 IMAGE-1 ... 19.95
CABLE, 500/2000 IMAGE-2 ... 19.95
CABLE, 6' MIDI LINK ... 9.95
CABLE, A1000 TO IMAGE-1 ... 19.95
CABLE, A1000 TO IMAGE-2 ... 19.95
CABLE, AMIGA TO NEC II ... 24.95
CABLE, AMIGA TO SONY 1302 ... 19.95
CABLE, AMIGEN TO 1084 ... 19.95
CABLE, DB25M DB25F 8 FT ... 19.95
CABLE, DB9M TO DB9F 6 FT. ... 16.95
CABLE, DRIVE EXT. 2 FT. ... 19.95
CABLE, DRIVE EXT. 3 FT ... 27.95
CABLE, FLCK FIX TO NEC II ... 19.95
CABLE, FLCK FIX-SONY 1302 ... 19.95
CABLE, M-CENTR TO M-CENTR ... 22.95
CABLE, MONITOR EXT 4FT ... 21.95
CABLE, MOUSE EXTENSION ... 14.95
CABLE, PARALLEL A1000 ... 19.00
CABLE, PARALLEL A500/2000 ... 19.00
CABLE, SCSI RIBBON, 2 FT. ... 15.95
CABLE, SERIAL A1000 ... 19.00
CABLE, SERIAL A500/A2000 ... 19.00
CLEANING KIT (ALLSOP) ... 14.95
CLEANING KIT SM. ALLSOP ... 6.95
CONTROL CENTER FOR A500 ... 71.95
COPY/CAMERA STAND, LIGHTS ... 64.95
CURTIS DIAMOND ... 33.95
CURTIS DIAMOND PLUS ... 46.95
CURTIS EMERALD ... 40.95
CURTIS PHONE LINE SURGE ... 12.36
CURTIS RUBY ... 58.95
CURTIS SAPPHIRE ... 56.95
DESKJET BLACK CART. ... 15.95
DESKJET COURIER FONT CART. ... 62.95
DESKJET DUST COVER ... 29.95
DESKJET EPSON FX-80 EMUL. ... 62.95
DESKJET HELV 8 & 10 PT. ... 79.95
DESKJET HELV ASCII FONTS ... 104.95
DESKJET LANDSCAPE FONT ... 79.95
DESKJET LANDSCAPE UPGRADE ... 79.95
DESKJET LETTER GOTHIC ... 79.95
DESKJET PRESENTATION FONT ... 79.95
DESKJET PRESTIGE ELITE ... 79.95
DESKJET RMN/HELV SOFT FNT ... 79.95

DESKJET STAND/ORGANIZER ... 84.95
DESKJET TMS RMN 8 & 10 PT ... 79.95
DESKJET TMS RMN ASCII ... 104.95
DIGI-DROID ... 69.95
DISK FILE 30 ... 7.95
DISK FILE 80 ... 19.95
DUST COVER FOR A500 ... 8.95
DUST COVER FOR A2000 ... 11.95
DUST COVER, A1000 W/MON. ... 13.95
EPYX 500XJ JOYSTICK ... 14.95
ERGO STICK ... 17.95
FLICKER MASTER ... 13.95
FLOPPY WALLET ... 7.95
FOUR PLAYER ADAPTER ... 11.95
• FUJI DISKETTES ... 19.95
GAME PLAY COMPACT DISK ... 12.95
GENDER BENDER (DIGI-VIEW) ... 20.95
KEYBOARD SKIN A2000 ... 19.95
KEYBOARD SKIN A500 ... 19.95
KEYBOARD SKIN A1000 ... 19.95
KRAFT MAZEMASTER JOYSTICK ... 9.95
LASERJET-2 TONER CART. ... 104.00
LASERJET-2 8.5 X 14 TRAY ... 104.95
LASERJET-2 COURIER FTS. ... 144.50
LASERJET-2 ENVELOPE TRAY ... 79.95
LASERJET-2 LASER CNTRL SW. ... 117.95
LASERJET-2 PLOTTER SOFTW. ... 117.95
LASERJET-2D ENVELOPE TRAY ... 279.95
LIGHT GUN ... 48.95
MICHTRON DISK FOLDER ... 10.95
• MIDI MAGIC ... 97.95
MIND LIGHT 7 ... 164.95
MOUSE COVER W/GLASSES ... 5.95
MOUSE MATS ... 6.95
MOUSE WASH ... 6.95
PAINTJET BLACK CART. ... 23.60
PAINTJET COLOR CART. ... 29.10

PAINTJET CUT-SHEET PAPER ... 19.10
PAINTJET DUST COVER ... 19.95
PAINTJET PAPER CATCHER ... 23.95
PAINTJET PRINTER STAND ... 66.95
PAINTJET TRANSPARENCY FLM ... 54.05
PAINTJET Z-FOLD PAPER ... 14.95
POWERPLAYERS JOYSTICK ... 19.95
PRINT & WEAR ... 13.95
QUICKSHOT II JOYSTICK ... 12.95
RAM, 1 MEG X 1-10 CHIPS ... 30.00
RAM, 1 MEG X 8-12 SIMMS ... 340.00
RAM, 256K X 1-10 CHIPS ... 12.00
RAM, 256K X 1-15 CHIPS ... 10.00
RAM, 256K X 8-12 SIMMS ... 88.00

PROGRAMMERS BOX:

LATTICE C 5.0 ... $195.00
LATTICE C++ ... $345.00
AREXX ... $ 32.95
W-SHELL ... $ 32.95
GFA BASIC 3.0 ... $ 90.95

RUGGED W. 16KB +4 FONTS! ... 124.95
• SONY DISKETTES, COLORED ... 22.95
SONY DISKS 10 PACK ... 20.95
TAC-2 JOYSTICK ... 10.95
TEAK-TECH 120 DISKHOLDER ... 32.95
TEAK-TECH 60 DISKHOLDER ... 27.95
TRACKBALL (2 BUTTON MOUSE) ... 39.95
TRANSVESTOR 1000 ... 23.95
TRANSVESTOR 2500 ... 41.95
WINNER JOYSTICK ... 14.95
XEROX 4020 4-PACK INK ... 19.95

FREE SHIPPING!*

CREATIVE COMPUTERS is both a mail order company with a store's support and three store showrooms with mail order prices. Visit one of our showrooms today. You'll be amazed!

3 RETAIL SHOWROOM LOCATIONS TO SERVE YOU

SANTA MONICA & WESTSIDE: 318 Wilshire Blvd., Santa Monica, CA 90401. Tues.-Sat. 11-7 p.m./Sun. 11-5 p.m./Phone: 213-394-7779

LAWNDALE & SOUTH BAY: 4453 Redondo Beach Blvd., Lawndale, CA 90260. Mon.-Sat. 11-7 p.m./Phone: 213-542-2292

VENTURA, OXNARD & SANTA BARBARA: 2112 E. Thompson Dr., Ventura, CA 93001. Tues.-Sat. 11-7 p.m./Sun. 12-5 p.m./Phone: 805-652-0325

COMMODORE AMIGA™

THE CREATIVE COMPUTERS ADVANTAGE:
- Authorized Amigo Dealer
- The largest dealer of Amiga products in the World
- Three store locations means excellent suppart
- Amigo-specific—unlike the competition, we don't claim being Amiga specific while selling other brands under onather business nome
- Authorized service center
- Uniform low pricing and largest selection, no hidden costs or catches
- We don't chorge your cord until the product ships

CUSTOMER SERVICE: Call 213-542-2292 from 11 a.m. to 7 p.m. PST for technical questions.

VISA · MasterCard · AMERICAN EXPRESS Card · DISCOVER

***SHIPPING INFO:** No surcharge for Visa/MosterCard; 4% surcharge for Americon Express; 1% surcharge for Discover. Shipping within the continentol United States for software orders over $100 (U.S.) is FREE. Please call for all other shipping information. Educational, corporate and aerospace purchase orders occepted.

RETURN POLICY: Coll Customer Service at (213) 542-2292 for return authorization. All returns without authorization number (RMA#) will be refused. Returned products must be in original condition and packaging. No refund on defective products or products that do not perform satisfactorily. We moke no guarantees for product performonce.

CONDITIONS: Creative Computers reserves the right to limit the sales of any items to local in-person pick-up only. Prices and availability subject to chonge without notice.

CREATIVE COMPUTERS GOES INTERNATIONAL!

INTERNATIONAL TOLL-FREE ORDERING POLICIES: Minimum order $100 U.S. Orders only please. Absolutely no product inquiries, or any shipping inquiries, will be taken over the tall-free phone lines. For any other information, please call (213) 542-2292. We have operators versed in the following languages: English, French, and Spanish, anly. Shipping prices vary and are quoted following your order. Payment is made via credit card or electronic fund transfer. For any other information please call aur regulor tall lines.

INTERNATIONAL PHONE NUMBERS: Canada 1-800-548-2512; Italy 1678-74086; United Kingdom 0800-89-1178; France 19-0590-1099; W. Germony 01-30-810466; Japan 0031-11-1351; Australia 0014-800-125-712; Netherlands 06-022-8613; Norway 050-12029; Denmark 0434-0297; Switzerland 046-05-3420.

WE CAN BEAT ANY ADVERTISED PRICE! BUT WE SELDOM HAVE TO — OUR PRICES ARE THE LOWEST! (IF YOU SEE A LOWER PRICE ON AN ITEM, GIVE US A CALL.)

Programming in M2Amiga

Spontaneous speech

by Anthony Bryant
Copyright © 1989 A. Bryant

Anthony Bryant is a computer engineer working with remote control systems and robotics. He programs the Amiga in his spare time, and has developed a number of general purpose Modula-2 routines. 'NarratorSupport' in this article is just one module from his library of I/O device support modules.

Perhaps it is because the designers of Amiga's software did not provide a ready-made routine for speaking (though they did provide a companion routine to Translate) that many programs written for the Amiga are speechless. This is in spite of the fact that the Amiga is and remains one of the few personal computers that comes with built-in speech capability.

NarratorSupport

The *NarratorSupport* (see *NarratorSupport.def* and *NarratorSupport.imp*) routines presented here are a quick and easy way to add speech to your existing M2Amiga programs. They are especially useful for adding spoken instructions to inform the user on how to get the most out of your program or how to extricate himself from an error condition.

Include the PrepareNarrator() procedure at the top of your program in order to open the narrator device. (Note: 'narrator.device' *must* be in the 'devs' directory of your boot disk). Opening the narrator device also automatically opens the audio device and initializes default parameters for the speaking voice. (see definition module *Narrator* for default parameters). If you don't like the default values for pitch, rate, volume, frequency, sex or mode, you can change them at any time, prior to speaking, with procedure SetVoiceParams(). Parameters submitted are checked to see if they are within range, so you can submit out of range parameters (for example (-1)) if you only want to change one or two of the values.

Procedure Say() is designed to interact with procedure Translate() (from the definition module *Translator*), so the syntax of unstructured strings, i.e. address and length, is the same. (Note: 'translator.library' *must* be in the 'libs' directory of your boot disk in order to use Translate()). Say() can also be used alone to speak phoneme codes that you provide. Refer to the ROM Kernal Manual for specifics pertaining to phoneme codes. In certain circumstances you may find it easier to roll your own phoneme codes, in order to get just the right emphasis or stress on the words. Use Say() with or without Translate() as often as you like within your program. You may find that Translate() gives a better rendition when you pass it purposely misspelled grammar - 'fractured fraises' - if you like. Finally, don't forget procedure CloseNarrator() to tidy things up at the conclusion of your program.

Say What?

This version of *NarratorSupport* uses a minimal implementation of Say(), taking advantage of the most commonly used 'synchronous' form of device I/O. When a voiceRequest is posted, using DoIO(), control does not return until the I/O request has completed. At that time the io_Error field of the I/O request block returns the request completion status. Only if the error is zero, does the speech actually commence. The most common error likely is the phonErr code (see definition module *Narrator* for error codes) in which case the io_Actual field contains the number of bytes processed up to the error and therefore contains the position in the phonemeBuffer that caused the halt. This is similar to the way Translate() returns a nonzero code, indicating that there is not enough space in the phonemeBuffer to hold the entire translation. In this case, Translate() breaks the translation at a grammatical word boundary and returns the position at which the translation ended.

Speak Easy

Assuming that speech has commenced, your program will want to be a good multitasking neighbor and sleep, or maybe print some text or do other processing. Be aware of the time it takes to actually complete speaking and that the time varies greatly with the rate parameter. Module *speech.mod* is a simple test program, demonstrating these routines in concert, and may be used as a template for experimentation or to check phrasing prior to including speech within a larger context.

So let's hear from you!

Listing 1: *NarratorSupport.def* - definition module

```
DEFINITION MODULE NarratorSupport;   (* in part *)
(* ---------------------------------------------------------------------------
 * BASED ON Narrator Device routines in ROM Kernal Manual by  Rob Peck
 *
 * M2Amiga Modula-2 adaptation by Anthony Bryant
 * ---------------------------------------------------------------------------*)
FROM SYSTEM IMPORT
 ADDRESS;
FROM Narrator IMPORT
 narratorName;


(* see DEFINITION MODULE Narrator for error codes and default voice params *)


PROCEDURE PrepareNarrator(): LONGINT;

PROCEDURE CloseNarrator();

PROCEDURE SetVoiceParams(rate, pitch, mode, sex, vol, freq: INTEGER);

PROCEDURE Say(outPhonemes: ADDRESS; outLen: LONGINT): LONGINT;


END NarratorSupport.def
```

Listing 2: *NarratorSupport.imp* - implementation module

```
IMPLEMENTATION MODULE NarratorSupport;  (* in part *)
(* ---------------------------------------------------------------------------
 * BASED ON Narrator Device routines in ROM Kernal Manual by  Rob Peck
 *
 * M2Amiga Modula-2 adaptation by Anthony Bryant
 * ---------------------------------------------------------------------------*)
FROM SYSTEM IMPORT
 ADR,ADDRESS,LONGSET;

FROM Narrator IMPORT
 narratorName,phonErr,male,female,natural,robotic,Narrator,NarratorPtr,
 minRate,maxRate,minPitch,maxPitch,minFreq,maxFreq,minVol,maxVol;

FROM Exec IMPORT
 UByte, MsgPortPtr,write,
 OpenDevice, CloseDevice, DoIO;

FROM ExecSupport IMPORT
 CreatePort,DeletePort,CreateExtIO,DeleteExtIO;


VAR  (* global *)
   voicePort: MsgPortPtr;
   voiceRequest: NarratorPtr;
   audChanMasks: ARRAY [0..3] OF UByte; (* channel masks *)


(* PrepareNarrator - Open Narrator Device and initialize a Narrator IORequest
 * block with defaults or if there was a problem, return error code (or zero
 * if success)   [see DEFINITION MODULE Narrator for possible error codes]
 *)
PROCEDURE PrepareNarrator(): LONGINT;
 VAR
   error: LONGINT;

 BEGIN
   error:= -1; (* assume failure ! *)

   voicePort:= CreatePort(0,0);
   IF (voicePort = NIL) THEN RETURN (error); END; (* error during CreatePort *)

   voiceRequest:= NarratorPtr(CreateExtIO(voicePort,SIZE(Narrator)));
   IF (voiceRequest = NIL) THEN
      DeletePort(voicePort); RETURN (error);      (* error during CreateExtIO *)
   END;
   (* now set up channel masks *)
   audChanMasks[0]:=  3; audChanMasks[1]:= 5;
   audChanMasks[2]:= 10; audChanMasks[3]:= 12;

   voiceRequest^.chMasks:= ADR(audChanMasks);
   voiceRequest^.nmMasks:= 4;

   (* open device and initialize with defaults - defPitch, defRate, etc *)

   OpenDevice(ADR(narratorName), 0, voiceRequest, LONGSET{});
   error:= LONGINT(voiceRequest^.message.error);

   IF (error # 0) THEN                          (* error during OpenDevice *)
   DeleteExtIO(voiceRequest);
   DeletePort(voicePort);
   END;

   RETURN (error);  (* should be zero, indicating success ! *)
 END PrepareNarrator;


(* CloseNarrator - Close Narrator Device and free structures.
 *)
PROCEDURE CloseNarrator();
 BEGIN

   IF (voiceRequest # NIL) THEN
       CloseDevice(voiceRequest);
       DeleteExtIO(voiceRequest);
   END;

   IF (voicePort # NIL) THEN
       DeletePort(voicePort);
   END;

 END CloseNarrator;


(* SetVoiceParams - change default parameters .. (-1) if no change
 *                  otherwise parameter is checked for min/max bounds
 *)
PROCEDURE SetVoiceParams(rate, pitch, mode, sex, vol, freq: INTEGER);
 BEGIN
  IF (voiceRequest # NIL) THEN (* proceed *)

   IF (rate >= minRate) AND (rate <= maxRate) THEN
      voiceRequest^.rate:= CARDINAL(rate); (* a new rate *)
   END;
   IF (pitch >= minPitch) AND (pitch <= maxPitch) THEN
      voiceRequest^.pitch:= CARDINAL(pitch);   (* a new pitch *)
   END;
   IF (mode = natural) OR (mode = robotic) THEN
      voiceRequest^.mode:= CARDINAL(mode);   (* a new mode *)
   END;
   IF (sex = male) OR (sex = female) THEN
      voiceRequest^.sex:= CARDINAL(sex); (* a new sex ? *)
   END;
   IF (vol >= minVol) AND (vol <= maxVol) THEN
      voiceRequest^.volume:= CARDINAL(vol); (* set a new volume *)
   END;
   IF (freq >= minFreq) AND (freq <= maxFreq) THEN
      voiceRequest^.sampFreq:= CARDINAL(freq); (* set a new sample frequency *)
   END;

  END; (* IF *)
 END SetVoiceParams;
```

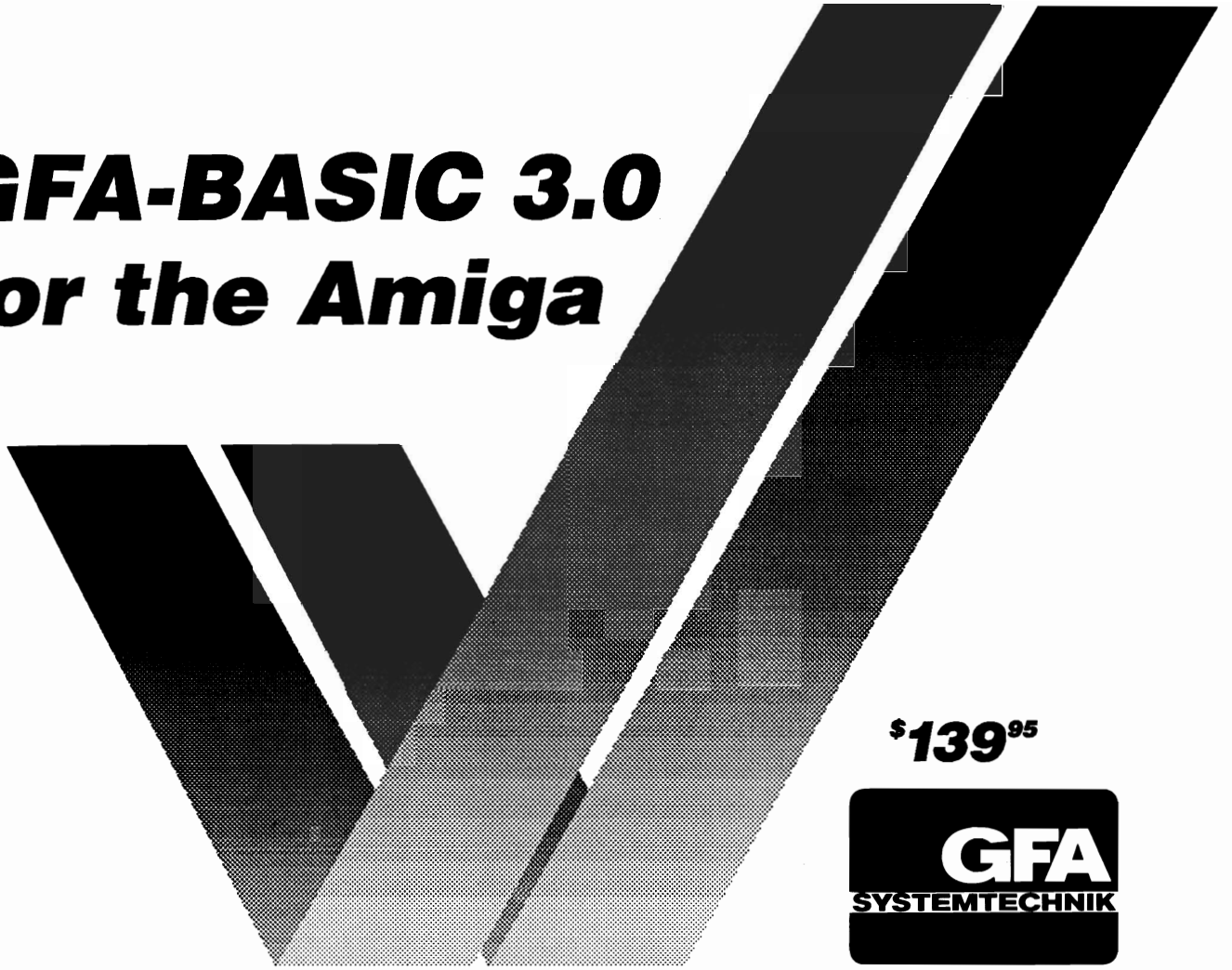
```
(* Say - after translation, say these phoneme codes
 *)
PROCEDURE Say(outPhonemes: ADDRESS; outLen: LONGINT): LONGINT;
 VAR
   error: LONGINT;

 BEGIN
 error:= -1; (* assume failure *)
 IF (voiceRequest # NIL) THEN (* proceed *)

   WITH voiceRequest^.message DO
     data:= ADDRESS(outPhonemes);  (* phonemes to say *)
     length:= LONGCARD(outLen);
     command:= write;  (* CMD_WRITE *)
   END;

   DoIO(voiceRequest);        (* post request to narrator device *)
   error:= LONGINT(voiceRequest^.message.error);

   (* check the io_Error field for a phoneme error (ND_PhonErr  -20) and
    * if so, then io_Actual field is the position where the error occured
    *)
   IF (error = phonErr) THEN
     error:= LONGINT(voiceRequest^.message.actual); (* send position instead *)
    END;

 END; (* IF *)

 RETURN (error);   (* should be zero, else check Narrator error code *)
 END Say;

END NarratorSupport.imp
```

Listing 3: *Speech.mod* - test program for speech routines

```
MODULE speech;   (* test the NarratorSupport routines *)

(* M2Amiga Modula-2 adaptation by Anthony Bryant *)

FROM SYSTEM IMPORT ADR;

FROM Arts IMPORT TermProcedure;

FROM Dos IMPORT Delay;

FROM NarratorSupport IMPORT
 PrepareNarrator, CloseNarrator, SetVoiceParams, Say;

FROM Translator IMPORT
 Translate;

FROM InOut IMPORT
 WriteLn, WriteString, WriteInt;



CONST
   phonemeBufLen = 256;
VAR
   phonemeBuffer: ARRAY [0..phonemeBufLen-1] OF CHAR;
   speechText: ARRAY [0..7],[0..79] OF CHAR;
   error: LONGINT;
   i: CARDINAL;

(* strlen - subroutine to regurgitate length of a null-terminated string
 *)
PROCEDURE strlen(text: ARRAY OF CHAR): LONGINT;
 VAR
   length: LONGINT;
 BEGIN
   length:= 0;
   WHILE (text[length] # 0C) DO INC(length); END;
   RETURN (length);
 END strlen;

BEGIN  (* here we go ! *)
   speechText[0]:= "The Amiga speech synthasis system is now activated.";
   speechText[1]:= "Listen, as you read the Transactor Amiga magazene.";
   speechText[2]:= "Narraytor Support is available to Modula 2 prowgrammers,";
   speechText[3]:= "Using defanytion modules, Narraytor, and, Translate, to speak.";
   speechText[4]:= "Proceedure, Translate, gets the phownetic spelling from this text.";
   speechText[5]:= "Proceedure, Say , produces sound from phowneme codes in the phoneme buffer.";
   speechText[6]:= "Use, proceedure, Set Voice Params, to change the voice style.";
   speechText[7]:= "See the Transactor Amiga magazene for details.  Cheers.";

   WriteString(" ** Speech Test ** ");  WriteLn;

   TermProcedure(CloseNarrator); (* leave a tidy Amiga *)

   error:= PrepareNarrator();
   IF (error # 0) THEN
     WriteString("   Narrator error !  ");  WriteInt(error,4);  WriteLn;
   END;

   (* parameters: rate,pitch,mode,sex,vol,freq *)
   SetVoiceParams(-1,220,-1,-1,-1,-1);  (* change pitch parameter *)

   WriteString("    Listen ... "); WriteLn;

   FOR i:= 0 TO 7 DO (* flip through the speech text *)
     error:= Translate(ADR(speechText[i]), strlen(speechText[i]),
                 ADR(phonemeBuffer), phonemeBufLen);
     IF (error # 0) THEN
       WriteString("   Translator error !  ");  WriteInt(error,4);  WriteLn;
     END;

     error:= Say(ADR(phonemeBuffer), strlen(phonemeBuffer));
     IF (error # 0) THEN
       WriteString("   Say error !  ");  WriteInt(error,4);  WriteLn;
     END;

     Delay(30); (* task sleeping while Amiga is speaking *)
   END;

   WriteString("    Done! "); WriteLn;

   (* TermProcedure() kicks in here *)
END speech.mod
```

. . . if you have at least 3 years of assembly language programming experience, preferably one year of 68000 or have an NTSC/PAL Video Hardware background. Join the team that makes no compromises and help us change the world. Key positions available at the fastest growing Amiga company. Hackers call Stephanie at 913-354-1146 or write to Stephanie at 115 West Crane Street, Topeka KS 66603.

GFA-BASIC 3.0 for the Amiga



$139.95

GFA SYSTEMTECHNIK

Boldly goes where no BASIC has gone before.

- High-Speed Interpreter for easy program development
- Over 300 powerful commands
- FAST! – Execution times comparable to C
- In-line C and Assembler Commands
- Easy access to all Amiga libraries
- Extensive Amiga commands with submenus and built-in file requester
- Built-in Text Editor with syntax checking, procedure hiding and auto-indenting
- 400-page comprehensive manual
- Includes Run-Time Interpreter

Available NOW

from

Antic Software

544 Second Street, San Francisco, CA 94107

Call today to order:

800-234-7001

or see your local Amiga dealer.

(GF9200)

GFA-BASIC 3.0 is a trademark of GFA Systemtechnik, Germany; Antic Software and Antic are trademarks of Antic Publishing, Inc.; Amiga is a registered trademark of Commodore-Amiga Inc.

Getting A Handle On File Systems

Part one of a series

by John Toebes and Doug Walker
Copyright © 1989 John A. Toebes, VIII and Douglas Walker

Doug Walker is a Senior Systems Developer at SAS Institute, Inc. in Cary, North Carolina. He has been programming the Amiga since 1986 as a member of The Software Distillery, which is responsible for many popular freely redistributable programs including Hack and Hack Lite, PopCLI, MemWatch and Blink. He can be reached on PeopleLink (DWALKER) and Usenet (mcnc!rti!sas!walker).

John Toebes is Manager of the Motorola Compiler Department at SAS Institute, Inc., responsible for the recent release of the Lattice C compiler. He is also coordinator for the Software Distillery. He can be reached on Bix (JTOEBES) and Plink (JTOEBES) as well as Usenet (mcnc!rti!sas!toebes).

What is a file system?

You use AmigaDOS whenever you deal with files on a floppy disk, hard disk or RAM disk. You call it to create, delete, rename, read and write files, among other operations. The DOS has to know lots of details about the media it is using: 3.5" floppy, 5.25" floppy, hard disk, and so forth. And if someone someday wanted to add WORM (Write Once, Read Many) optical disks, networking, Fast Filing Systems and so on, DOS would have to know how to access them as well.

How can AmigaDOS know so much about so many different pieces of hardware, some of which were developed after AmigaDOS itself? The answer is simple: modularization. While it does have its flaws, AmigaDOS is one of the most modular and flexible disk-accessing systems on any microcomputer.

AmigaDOS file access is managed using the concept of *devices*. DF0: is a device that accesses your internal floppy drive; SER: is a device that accesses the serial port, and so forth. Each device has a program associated with it that takes requests from AmigaDOS and does whatever is necessary to see the requests are filled. Taking advantage of the multitasking nature of the Amiga, these programs are all separate processes, and are running at all times after the corresponding device is initialized. Any number of devices may be active at one time, and you can install custom devices of your own quite easily.

The program associated with the device can be relatively simple or it can be quite complicated. For SER:, PAR: and PRT:, the program doesn't have to know anything about dealing with files. All it needs to do is read and write data. A simple device-driver of this type is called a *handler*. A handler merely accepts a stream of data and routes it to the correct I/O device, and accepts return data and routes it to the application using it.

Handlers are fine for dealing with SER:, PAR: and PRT:, which have no concept of files and directories, but they fall short of the mark when we look at some of the other devices. Most Amiga owners, for example, would not be happy with a hard disk device that didn't allow creating or deleting files! To deal with these more complicated devices, a more complicated program is needed. This more complicated program is called a *file system*.

Slightly less complicated than a file system is the special type of handler called a *console handler*, such as 'CON:' or 'RAW:'. There are several packet types that are supported only by this type of handler. An example of such a packet is the ACTION_RAW_MODE, which switches a console in and out of 'raw' mode.

It is a file system that gives Amiga disks their hierarchical structure, with directories containing subdirectories containing subdirectories ad infinitum. The file system makes sure that two programs don't end up writing to the same file at the same time, that finds room on the media it is managing for additional data, that frees disk space when files are deleted, and in sum does whatever housekeeping is needed to allow you to access the device.

In general, a file system will have to manage a *volume* - normally some form of storage medium, either removable or fixed - giving us two ways to reference a handler. One is by its own name, such as 'DF0:' or 'FD1:'. The other is by the volume's name, such as 'My_Backup_Disk:' or 'Workbench1.3:'. In both cases, AmigaDOS finds the target handler by searching the system-wide device list for an entry of the given name. The device entry was created by AmigaDOS when the handler was initialized; additional volume entries are added by the

handler itself as volumes are encountered. This is important to note, since many of the requests to the handler reference a volume node.

Communicating with file systems

How do we communicate with a file system or handler? As you might expect, it is done in the same way as most interprocess communication on the Amiga: with messages and message ports. Each handler or file system process has a message port set up waiting for requests to come in. You can find the address of the message port for the device you are interested in with the *DeviceProc* function. The call

```
port = DeviceProc("DF0:");
```

will put the address of the message port for the internal floppy drive into the variable 'port'.

Having located the message port for the handler, we must now format our requests in such a way that the handler or file system will understand them. A special structure called a *StandardPacket* is used to communicate with AmigaDOS devices. It is defined in the C include file <libraries/dosextens.h> and is reprinted here as Figure 1. The actual action requests that we can pass to the handler will be detailed later on.

The *StandardPacket* structure contains two members: *sp_Msg*, which is the message structure used to route the packet to the file system; and *sp_Pkt*, which is the actual data to be sent. Some fields in both the *sp_Msg* and the *sp_Pkt* members must be initialized to tell DOS where to find the packet data and where to send the reply.

The field *sp_Msg.mn_Node.ln_Name* must be set to point to the *sp_Pkt* structure. The *sp_Pkt.dp_Link* field of the *DosPacket* structure, in turn, must be set to point back to the *sp_Msg* structure. You must create a message port (with the *CreatePort* function) for DOS to send the reply to, and initialize the *sp_Pkt.dp_Port* field with the address of that port. Note that the *sp_Msg.mp_ReplyPort* field is ignored. This is a quirk of the DOS message-passing mechanism. Instead of replying to the reply port specified in the message structure, DOS replies to the port specified in the packet structure. Before it does so, DOS resets the field to point to its own message port rather than yours, so you must set the *sp_Pkt.dp_Port* field to your message port every time you send a packet to a DOS device.

Once the packet overhead is done, we must fill in the meat of the packet. The *dp_Type* field tells AmigaDOS what you want it to do. Depending on what type (or *action*) you are using, you may need to fill in one or more of the *dp_Arg* fields with additional data. After the packet is returned to you, AmigaDOS will have set the *dp_Res* fields to let you know what happened to your request.

The simple C function listed at the end of this article initializes a DOS packet, allocates a reply port, sends the packet to DOS, waits for a reply and returns to the caller. Although not very efficient for repeated calls (you must reallocate the reply port every call!), it is a good example and quite easy to use.

```
struct DosPacket {
   struct Message *dp_Link;   /* EXEC message        */
   struct MsgPort *dp_Port;   /* Reply port for the packet */
                              /* Must be filled in each send. */
   LONG dp_Type;              /* See ACTION_... below and
                               * 'R' means Read, 'W' means Write to the
                               * file system */
   LONG dp_Res1;              /* For file system calls this is the result
                               * that would have been returned by the
                               * function, e.g. Write ('W') returns
                               * actual length written */
   LONG dp_Res2;              /* For file system calls this is what would
                               * have been returned by IoErr() */

/*  Device packets common equivalents */
#define dp_Action  dp_Type
#define dp_Status  dp_Res1
#define dp_Status2 dp_Res2
#define dp_BufAddr dp_Arg1

   LONG dp_Arg1;
   LONG dp_Arg2;
   LONG dp_Arg3;
   LONG dp_Arg4;
   LONG dp_Arg5;
   LONG dp_Arg6;
   LONG dp_Arg7;
}; /* DosPacket */

/* A Packet does not require the Message to be before it in memory, but
 * for convenience it is useful to associate the two.
 * Also see the function init_std_pkt for initializing this structure */

struct StandardPacket {
   struct Message   sp_Msg;
   struct DosPacket sp_Pkt;
}; /* StandardPacket */
```

Figure 1: The DosPacket and StandardPacket structures, used to communicate with DOS devices

Packets that are sent to a file system can be divided into five basic functional categories:

1) Object/File Manipulation
   These actions generally deal with accessing the data contained within the objects (files) controlled by the file system.

2) Directory Manipulation/Information
   These actions are used to gain access to and manipulate the high-level structure of the file system.

3) Volume Manipulation/Information
   The few actions in this category allow access to the specific volume controlled by the file system.

4) Handler Maintenance and Control
   These allow control over the file system itself, independent of the actual volume or structure underneath.

5) Handler Internal
These actions are never sent to the handler directly. Instead they are generally responses to requests made by the handler. The handler makes these responses look like requests in order to simplify processing.

In addition, there are several packet types that are documented within the system include files that are now obsolete. A file system is not expected to handle these packets; any program that sends these packets cannot expect them to work:

ACTION_DISK_CHANGE	**33**
ACTION_DISK_TYPE	**32**
ACTION_EVENT	**6**
ACTION_GET_BLOCK	**2**
ACTION_SET_MAP	**4**

BPTRS and BADDRs

A DOS packet will require a BCPL pointer address. BCPL is a parent language of the C language in use today, but BCPL pointers differ from C pointers and from assembler addresses in that they can only point to a (four-byte) longword, not an individual byte. A BCPL pointer with the value 0x0000001 is thus actually pointing to byte 0x00000004 in the machine (which, as you may know, is AbsExecBase, the one fixed address on the Amiga). This method allows you to address four times more memory than you otherwise could, but at the cost of the ability to point to specific bytes. This was quite important in the days when addresses were limited to 16 bits, since it extended your addressable memory from 64K to 256K.

Although we have 32-bit addresses on the Amiga, the original AmigaDOS handlers were written in BCPL, and the AmigaDOS packet interface reflects this. If we changed the packet fields to take normal addresses, all existing Amiga programs would need to be rewritten and replaced. Since this is not a reasonable option, we have to be able to convert back and forth between BCPL pointers (BPTRs) and C pointers. Two macros do the trick here: BADDR and MKBADDR.

```
#define BADDR(x)      ((APTR)((long)(x) << 2))
#define MKBADDR(x)    ((BPTR)((long)(x) >> 2))
```

If we have a normal C address and need a BCPL address, we call MKBADDR:

```
bptr = MKBADDR(aptr);
```

If we have a BCPL address and need a C address, we call BADDR:

```
aptr = BADDR(bptr);
```

Of particular note here is ACTION_DISK_CHANGE. The current *DiskChange* command sends a sequence of ACTION_INHIBITS to the handler to accomplish the task.

The remaining packet types are used only for console handlers and do not need to be implemented by a file system. Note that the last five of these packets are private packets used by ConMan and introduced by Bill Hawes:

ACTION_SCREEN_MODE 994
Switch the console between RAW and editing mode.

ACTION_WAIT_CHAR 20
Perform a timed read of a character. This implements the *WaitForChar* function.

ACTION_DOUBLE 2000
Perform a double open on a file handle.

ACTION_FORCE 2001
Force input into the handler as if typed from the keyboard.

ACTION_STACK 2002
Stack a FIFO line available for reading but not for display.

ACTION_QUEUE 2003
Queue a LIFO line available for reading but not for display.

ACTION_DROP 2004
Remove all queued and stacked lines from the handler.

Much of this information presented here can be located by cross-referencing information in the Developer Conference notes with the Bantam Books *AmigaDOS Manual*, as well as bits and pieces on the various Fred Fish disks.

However, because there is no single complete reference to these packet types, we attempt to present a consolidated view of all the packets here. Several structures are referenced here which can be found by looking at the include files <libraries/dos.h> and <libraries/dosextens.h>. You should become familiar with these files before attempting to work with a file handler.

For each of the packet types, we list the action name along with its corresponding number and any AmigaDOS routine that issues it. Also included is a list of each parameter, along with its type, as follows:

BPTR The address of the given object shifted right by 2.

LOCK A BPTR to a FileLock structure returned by a previous ACTION_LOCATE_OBJECT. A lock of 0 is legal, and indicates the root of the file system for the handler.

BSTR A BPTR to a string where the first byte indicates the number of characters in the string. This length byte is unsigned but, because it is stored in a byte, the strings are limited to 255 characters in length.

BOOL A 32-bit boolean value containing either DOSTRUE (-1) or DOSFALSE (0).

CODE A 32-bit error code as defined in the <libraries/dos.h> include file; only codes described there should be returned.

ARG1 The field *FileHandle->fh_Arg1*.

LONG Any arbitrary 32-bit value.

Object/file manipulation actions

These are supported by both handlers and file systems. In this way, an application can get stream-level access to both devices and files. One difference that arises is that a handler will not necessarily support an ACTION_SEEK, while it is generally expected for a file system to do so.

These actions work with a *FileHandle* structure that is filled in by a file-open in one of the three access modes:

```
ACTION_FINDINPUT     1005 Open(..., MODE_INPUT)
ACTION_FINDOUTPUT    1006 Open(..., MODE_OUTPUT)
ACTION_FINDUPDATE    1004 Open(..., MODE_UPDATE)
   ARG1: BPTR - FileHandle to fill in
   ARG2: LOCK - Location that name is relative to
   ARG3: BSTR - Name of file to be opened

   RES1: BOOL - Success/Failure
   RES2: CODE - Failure code if RES1 is DOSFALSE
```

All three actions will use the LOCK as a base location for the file to be opened. If a LOCK of 0 is passed, it is assumed that the file name given is relative to the root of the volume. Because of this feature, file names are not limited to a single name, but can instead be qualified with a colon and multiple slashes, allowing the file system to fully resolve the name. This eliminates the need for AmigaDOS or the application to parse names before sending them to the file system. Note, however, that the LOCK given *must* be associated with the file system. It is illegal to provide a LOCK taken from another file system.

The *FileHandle* is owned by the calling program (in the case of a call to *Open*, AmigaDOS allocates it automatically and then frees it upon a *Close*), and must be initialized before calling the file system. All fields must be zero except the *fh_Pos* and *fh_End* fields, which should be set to -1. Upon a successful open, the file system is responsible for filling in the *fh_Type* field to point to the file system process. Lastly, it must initialize *fh_Arg1* with something that allows it to uniquely locate the file. It is this field that is passed to the READ, WRITE, SEEK, END and TRUNCATE operations, and not the file handle itself.

FINDINPUT and FINDUPDATE are similar in that they will only succeed if the file already exists. FINDINPUT will open the file with a shared lock, while FINDUPDATE will open it with an exclusive lock. FINDOUTPUT will always open the file (truncating any existing one) with an exclusive lock.

```
ACTION_READ   'R'   Read(...)
   ARG1: ARG1 - fh_Arg1 field of the opened FileHandle
   ARG2: APTR - Buffer to put data into
   ARG3: LONG - Number of bytes to read

   RES1: LONG - Number of bytes read.
                0 indicates EOF; -1 indicates ERROR.
   RES2: CODE - Failure code if RES1 is -1
```

This action extracts data from the file at the current position. If fewer bytes remain in the file than requested, only those bytes remaining will be returned, with the appropriate value stored in RES1.

Any error is indicated by a -1 in RES1 and the return code in RES2. In case of any failure, the current file position remains unchanged so that the operation may be retried.

```
ACTION_WRITE  'W'   Write(...)
   ARG1: ARG1 - fh_Arg1 field of the opened FileHandle
   ARG2: APTR - Buffer to write
   ARG3: LONG - Number of bytes to write

   RES1: LONG - Number of bytes written.
   RES2: CODE - Failure code if RES1 not equal to ARG3
```

This action puts data into the file at the current postion. If the end of the file is passed, the file is automatically extended. Any failure is indicated by returning a RES1 that is not the same as the number of bytes requested, with the failure code in RES2.

In the case of a failure, the current file position is not updated (although the file may have been extended and some data overwritten) so that the operation may be safely retried.

```
ACTION_SEEK   1008  Seek(...)
   ARG1: ARG1 - fh_Arg1 field of the opened FileHandle
   ARG2: LONG - New Position
   ARG3: LONG - Mode: OFFSET_BEGINNING,
                OFFSET_CURRENT, or OFFSET_END

   RES1: LONG - Old Position. -1 indicates an error
   RES2: CODE - Failure code if RES1 = -1
```

ACTION_SEEK sets the current position in the file according to the mode and position. The previous seek position is returned. Any attempt to seek past the end of the file sets the current position to the end of the file.

```
ACTION_END 1007  Close(...)
   ARG1: ARG1 - fh_Arg1 field of the opened FileHandle

   RES1: LONG - DOSTRUE
```

This terminates all actions on an open file. It will return a DOSTRUE, since there is little an application can do to recover from a file-closing failure.

```
ACTION_TRUNCATE 1022 <sendpkt only>
   ARG1: ARG1 - fh_Arg1 field of the opened FileHandle

   RES1: BOOL - Success/failure
   RES2: CODE - Failure code if RES1 = DOSFALSE
```

This action is not supported by existing file systems, although new file system handlers are expected to support it. It will truncate an open file to the current seek position.

Directory manipulation actions

These are used for performing maintenance on the high-level structure of the file system on the media. In general they deal with directories and the names within those directories, although there is no requirement that any given file system support subdirectories.

```
ACTION_LOCATE_OBJECT 8 Lock(...)
   ARG1: LOCK - Lock to which name is relative
   ARG2: BSTR - Name of object to lock
   ARG3: LONG - Mode: ACCESS_READ/SHARED_LOCK,
                ACCESS_WRITE/EXCLUSIVE_LOCK

   RES1: LOCK - Lock on object; 0 indicates failure
   RES2: CODE - Failure code if RES1 = 0
```

This action implements the *Lock* functionality. Given a lock as a basis (with 0 indicating the root of the volume) and a name for the object, it will locate the object within the file system and create a *FileLock* structure associated with the object. The memory for this structure is maintained by the handler and freed by an ACTION_FREE_LOCK. Although it is not a requirement, if a handler expects to support the *Format* command with 1.3 and earlier releases, it must accept any illegal mode as ACCESS_READ.

An EXCLUSIVE_LOCK may only be created if there are no other outstanding locks on the given object and, once created, prevents any other locks from being created for that object. In general, the *key* field is used to uniquely identify the object.

In the *FileLock* structure returned, *fl_Volume* is expected to point to the volume entry in the DOS device list associated with the created lock. In addition, there are several diagnostic programs that expect all locks for a volume to be chained together off the *dl_LockList* field in the volume entry. It is recommended, though not required, that you follow this convention.

```
ACTION_COPY_DIR 19 DupLock(...)
   ARG1: LOCK - Lock to duplicate

   RES1: LOCK - Duplicated Lock; 0 indicates failure
   RES2: CODE - Failure code if RES1 = 0
```

This action's name is misleading - it doesn't manipulate directories. In fact, it implements the *DupLock* function by creating copies of any shared locks. These copies are subsequently freed with an ACTION_FREE_LOCK. Note that it is valid for a lock of 0 to be passed. In this case the duplicated lock will be non-zero.

```
ACTION_FREE_LOCK15 UnLock(...)
   ARG1: LOCK - Lock to free

   RES1: TRUE
```

This action implements the *UnLock* function by freeing the lock passed to it. It is acceptable to get a lock of 0 here, which should simply be ignored.

```
ACTION_EXAMINE_OBJECT 23 Examine(...)
   ARG1: LOCK - Lock of object to examine
   ARG2: BPTR - FileInfoBlock to fill in

   RES1: BOOL - Success/failure
   RES2: CODE - Failure code if RES1 = DOSFALSE
```

This action implements the *Examine* function by filling in the *FileInfoBlock* structure with information about the locked object. This routine is actually used for two different types of operations. Often it is simply called to obtain information about a given object. In other cases, it is called to prepare for a sequence of EXAMINE_NEXT operations in order to traverse a directory.

This seemingly simple operation is not without its quirks that the handler must be aware of. In particular, the *fib_Comment* field used to be 116 characters; however, some structures had only allocated 80 bytes it. In 1.2 and beyond, therefore, this field is limited to 80 bytes, with the remaining 36 bytes specified as reserved.

Similarly, although the *fib_FileName* field has room for 108 characters, only 30 are used. Lastly, both the *fib_EntryType* and the *fib_DirEntryType* fields must be set to the same value, since some programs look at one field while others look at the other.

```
ACTION_EXAMINE_NEXT   24 ExNext(...)
   ARG1: LOCK - Lock
   ARG2: BPTR - BPTR FileInfoBlock
   RES1: BOOL - Success/failure
   RES2: CODE - Failure code if RES1 = DOSFALSE
```

This action implements the *ExNext* operation, which is used to obtain all the members of a directory. In general, an application will call ACTION_EXAMINE, followed by a series of calls to ACTION_EXAMINE_NEXT until an ERROR_NO_MORE_ENTRIES failure code is returned.

If this were the only way it were used, this routine would not be all that complicated. However, among the situations that ACTION_EXAMINE_NEXT must handle are:

1) The application calls ACTION_EXAMINE_NEXT one or more times then stops invoking it before encountering the end of the directory.

2) The *FileInfoBlock* passed to ACTION_EXAMINE_NEXT is not always the same as the one passed to ACTION_EXAMINE, or even the previous EXAMINE_NEXT operation. Instead it is a copy of the *FileInfoBlock*, with only the *fib_DiskKey* and the first 30 bytes of the *fib_FileName* fields copied over.

3) Between ACTION_EXAMINE_NEXT operations, a request to delete the file just examined can be issued. In fact, any amount of time or number of requests can occur between these operations. ACTION_EXAMINE_NEXT must protect itself in these cases.

4) The lock passed to ACTION_EXAMINE_NEXT is not always the same as that used for previous operations. It is, however, a lock on the same object.

Because of these problems, ACTION_EXAMINE_NEXT is probably the trickiest action to write in any handler. Failure to cope with the above cases can be quite disastrous.

```
ACTION_CREATE_DIR  22 CreateDir(...)
   ARG1: LOCK - Lock to which name is relative
   ARG2: BSTR - Name of new directory to create

   RES1: BOOL - Success/failure
   RES2: CODE - Failure code if RES1 = DOSFALSE

ACTION_DELETE_OBJECT  16 DeleteFile(...)
   ARG1: LOCK - Lock to which name is relative
   ARG2: BSTR - Name of object to delete

   RES1: BOOL - Success/failure
   RES2: CODE - Failure code if RES1 = DOSFALSE

ACTION_RENAME_OBJECT  17 Rename(...)
   ARG1: LOCK - Lock to which the name is relative
   ARG2: BSTR - Name of object to rename
   ARG3: LOCK - Lock associated with target directory
   ARG4: BSTR - Requested new name for the object

   RES1: BOOL - Success/failure
   RES2: CODE - Failure code if RES1 = DOSFALSE
```

These three actions implement the high-level operations equivalent to the AmigaDos commands *MakeDir*, *Delete*, and *Rename* for single files. They take as their parameters a lock describing where the file is, and a name relative to that lock. It is the file system's responsibility to ensure that the operation is safe. In particular, the RENAME_OBJECT action allows files to be moved across directory bounds, and so must check that hidden directory loops are not created. Such problems are not always detected by the current floppy disk file system, allowing drawers on a volume to seemingly vanish leaving unrecovered space. It is also legal to do an ACTION_RENAME_OBJECT and specify the same name for an object but change the case.

For directory objects, the DELETE_OBJECT action must ensure that the directory is empty before allowing the operation.

```
ACTION_PARENT 29 Parent(...)
   ARG1: LOCK - Lock on object to get the parent of

   RES1: LOCK - Parent Lock
   RES2: CODE - Failure code if RES1 = 0
```

This action takes a lock on an object and creates a shared lock on the parent object. If the original object has no parent, a lock of 0 is returned. This operation is typically used in the process of constructing the fully qualified pathname of a given object.

```
ACTION_SET_PROTECT 21 SetProtection(...)
   ARG1:      - Unused
   ARG2: LOCK - Lock to which name is relative
   ARG3: BSTR - Name of object
   ARG4: LONG - Mask of new protection bits

   RES1: BOOL - Success/failure
   RES2: CODE - Failure code if RES1 = DOSFALSE
```

This action allows the application to modify the protection bits on an object. Note that the last four bits (RWED) are reversed in meaning, with a one-bit indicating that the operation is not allowed. By default, all files are created with the RWED bits set and all others cleared. Additionally, any action that modifies a file is required to clear the A (archive) bit.

```
ACTION_SET_COMMENT 28 SetComment(...)
   ARG1:      - Unused
   ARG2: LOCK - Lock to which name is relative
   ARG3: BSTR - Name of object
   ARG4: BSTR - New Comment string

   RES1: BOOL - Success/failure
   RES2: CODE - Failure code if RES1 = DOSFALSE
```

This allows the application to set the comment string on an object. If the object does not exist, DOSFALSE will be returned in RES1 with the failure code in RES2. The comment is limited to 79 characters, since that is all that will fit into a *FileInfoBlock*.

```
ACTION_SET_DATE 34 <sendpkt only>
   ARG1: LOCK - Object
   ARG2: BPTR - DateStamp

   RES1: BOOL - Success/failure
   RES2: CODE - Failure code if RES1 = DOSFALSE
```

This action allows the application to set the creation date on a locked object. It is also a source of some confusion, with some handlers implementing ARG2 as an APTR instead of a BPTR. Note as well that the parameter is a lock on the object, not a lock and name as found in the other packet types.

Volume manipulation actions

These actions are used to allow access to the underlying volume currently being manipulated by the file system.

```
ACTION_CURRENT_VOLUME 7  <sendpkt only>
   RES1: BPTR - Ptr to volume node of current volume
```

This returns a pointer to the volume node associated with the file system. There is no lifetime associated with this pointer, since the volume node may be removed from the device list when the file system mounts a different volume (such as when directed to by an ACTION_INHIBIT). This action is generally used by AmigaDOS to provide the volume line of a requester.

```
ACTION_DISK_INFO25 Info(...)
   ARG1: BPTR - Ptr to an InfoData struct to fill in
   RES1: BOOL - Success/Failure

ACTION_INFO  26 <sendpkt only>
   ARG1: LOCK - Lock
   ARG2: BPTR - Ptr to an InfoData Struct to fill in
   RES1: BOOL - Success/Failure
```

These are used to get information about the device and status of the file handler. ACTION_INFO appears only to be used by Workbench to detect when a volume is removed or inserted. ACTION_DISK_INFO is used by *Info* to report the volume status. These actions are generally expected to return DOSTRUE.

```
ACTION_RENAME_DISK 9  <sendpkt only>
   ARG1: BSTR - New disk name
   RES1: BOOL - Success/Failure
```

This action allows an application to change the name of the current volume. The file system must also remember to change the name stored in the volume node of the device list.

Handler control actions

A number of packets are defined to give the application control over how the file system is to work:

```
ACTION_DIE 5  <sendpkt only>
   RES1: BOOL - DOSTRUE
```

All new handlers are expected to respect this packet by switching into as quiescent a state as possible. Because of outstanding locks and the fact that the handler address is available to the world through the *DeviceProc* routine, it is unlikely that the handler can disappear completely. What it *can* do is release as many resources as possible and simply return an error on all packets sent to it.

```
ACTION_FLUSH  27 <sendpkt only>
   RES1: BOOL - DOSTRUE
```

This action causes the file system to flush out all buffers to disk to ensure that the underlying media is in a complete state *before* returning this packet. Note that if any writes are pending, they must be processed before responding. The packet is primarily used to allow an application to ensure that data are checkpointed against any power failure or system crash.

```
ACTION_MORE_CACHE  18 <sendpkt only>
   ARG1: LONG - Number of Blocks

   RES1: BOOL - DOSTRUE
   RES2: LONG - New number of cache blocks
```

This action allows changing of the number of internal buffers used by the file system for caching. A positive number increases the number of caches, while a negative number decreases the number of caches. In all cases, the number of current cache blocks are returned in RES2. This allows an application to find out the number of cache blocks by sending in a value of 0 (resulting in no change).

```
ACTION_INHIBIT  31 <sendpkt only>
   ARG1: BOOL - DOSTRUE=inhibit, DOSFALSE=uninhibit
   RES1: BOOL - Success/failure
```

This action is probably one of the most dangerous that a file system has to handle. When inhibited, the file system must not access any underlying media, and must return an error code on all attempts to access the device. Once uninhibited, the file system must assume that the media has been changed. In this light, the file system must have already flushed the buffers before the action is requested. If not, the handler is responsible for putting up a requester demanding the disk before returning the inhibit. Moreover, if any objects are open for writing, the handler may fail the inhibit request.

```
ACTION_WRITE_PROTECT  1023  <sendpkt only>
   ARG1: BOOL - DOSTRUE/DOSFALSE
   ARG2: LONG - 32 Bit passkey
   RES1: BOOL - DOSTRUE/DOSFALSE
```

This is a new packet defined for the Fast File System that allows a soft setting of the write protect flag. It is primarily intended for support of non-removable media like hard disks. ARG1 controls the setting of the bit with an optional passkey. If a passkey of 0 is given, then no passkey is necessary for unlocking the disk. Any other passkey given for locking the volume must be matched exactly on the unlock request.

```
ACTION_NETWORK_HELLO  2012  <sendpkt only>
   ARG1: BSTR - name of node
```

This private packet was introduced for a network handler to allow remote networks to install a node name on a given machine.

```
ACTION_HANDLER_DEBUG  2010  <sendpkt only>
   ARG1: LONG - Debug flags. 0 means no debugging.
   ARG2: ???? - Handler Private based on ARG1

   RES1: BOOL - Success/Failure
```

This action is a private packet used by all of the Software Distillery handlers to allow a debugging hook into the handler. By definition, the actual flags are private, with a value of 0 indicating no debugging is to be done.

```
ACTION_SET_TRANS_TYPE 2011 <sendpkt only>
   ARG1 - BSTR - Name of files to translate
   ARG2 - LONG - type of translation to apply
```

This private packet was introduced for an MS-DOS handler to allow control of which files are to be translated. Because MS-DOS text files have a CR/LF at the end of each line and AmigaDOS files only have an LF, it is necessary to perform translation of some files and not of others. By definition, a value of 0 indicates no translation (the default for all handlers) and any other value is handler-specific. The name format is also handler-specific; NULL is used to set a handler-wide translation default.

Handler-internal actions

Lastly, we have several actions that are generally used by handlers to allow returned messages from requested services to look like incoming request packets.

This allows the handler to request an asynchronous operation but be notified of the completion. The actual arguments are often ignored and these actions are *not* replied to.

```
ACTION_NIL 0  <internal>
```

Although not specifically an action, many returns look like this value because the action field has not been filled in.

```
ACTION_READ_RETURN 1001  <internal>
```

This is generally used to indicate the completion of an asynchronous read request.

```
ACTION_WRITE_RETURN   1002 <internal>
```

This is generally used to indicate the completion of an asynchronous write request.

```
ACTION_TIMER  30 <internal>
```

This action is used to indicate the passage of a time interval. Many handlers have a steady stream of ACTION_TIMER packets so that they can schedule house-keeping and flush buffers when no activity has occurred for a given time interval.

* * * * *

The list of AmigaDOS packet types we have presented here should be complete up to the present time. It is certain that a few new packets will be added with the introduction of 1.4, but these are likely to be upward extensions.

And that just about wraps up everything you wanted to know about communicating to a file system. Next issue we will use this information to actually write a simple file system that reads and writes Amiga-format diskettes. With this as an example we can proceed to look at alien handlers such as a NetWork handler, or even other disk formats.

Listing 1: Simple packet sending in C

```c
#include <exec/types.h>
#include <exec/memory.h>
#include <libraries/dosextens.h>

/*********************************************************************/
/* This function sends a packet to a DOS device and waits for a reply. */
/* It returns the primary result (dp_Res1) and, if supplied an array   */
/* of longs, will set the array values to indicate both results.      */
/*********************************************************************/

LONG sendpkt(pid,action,args,nargs,res)
struct MsgPort *pid;  /* process identifier... (handler's message port) */
LONG action,          /* packet type... (what you want handler to do)   */
     args[],          /* a pointer to a argument list            */
     nargs,           /* number of arguments in list             */
     res[2];          /* pointer to 2 longs for result, or NULL  */
                      /* if you are not interested in the result */
  {
  struct MsgPort        *replyport;
  struct StandardPacket *packet;
  LONG  count, lres, *pargs;

  if(res)
  {
     /* Set the results to 0 in case we quit early */
     res[0] = res[1] = 0L;
  }

  /* Create a message port for DOS to reply to */
  replyport = (struct MsgPort *) CreatePort(NULL,0);
  if(!replyport) return(0L);

  packet = (struct StandardPacket *)
     AllocMem((long)sizeof(struct StandardPacket),MEMF_PUBLIC|MEMF_CLEAR);
  if(!packet)
  {
     DeletePort(replyport);
     return(0L);
  }

  /* Set up linkage information                                  */
  /* Note that if you want to send multiple packets using the same  */
  /* StandardPacket struct, you can dispense with the first two     */
  /* assignments done here, but you MUST assign dp_Port every time! */
  packet->sp_Msg.mn_Node.ln_Name = (char *)&(packet->sp_Pkt);
  packet->sp_Pkt.dp_Link         = &(packet->sp_Msg);
  packet->sp_Pkt.dp_Port         = replyport;

  /* Let DOS know what we want it to do */
  packet->sp_Pkt.dp_Type         = action;

  /* Copy the args into the packet */
  pargs = &(packet->sp_Pkt.dp_Arg1);       /* address of first argument */
  for(count=0;count < nargs;count++)
     pargs[count]=args[count];

  PutMsg(pid,(struct Message *)packet); /* send packet */

  WaitPort(replyport);
  GetMsg(replyport);

  /* Get the primary result */
  lres = res[0] = packet->sp_Pkt.dp_Res1;

  /* if they supplied a result array, give them the results */
  if(res)
  {
     res[0] = lres;
     res[1] = packet->sp_Pkt.dp_Res2;
  }

  /* Clean up */
  FreeMem((char *)packet,(long)sizeof(struct StandardPacket));
  DeletePort(replyport);

  return(lres);
}
```

DOS Shells: No More CLI!

Four shells: Dillon's, Tshell, Wshell, and DOS 1.3

by Mike Hubbartt

Soon after I got my Amiga, a friend helped me set up Matt Dillon's shell... for the additional control it gave over the machine, he explained. The plain CLI lacked features that made it easy to use. The disk drive was constantly being accessed to load in commands; it was monotonous to repeatedly type in a command line when only a small part of it needed changing each time. The AmigaDOS commands weren't the easiest things to learn, although the documentation from the ARP commands helped until I bought the AmigaDOS manual.

The built-in commands in Dillon's shell saved disk access time, and I could get a list of all the commands just by pressing the help key! After looking in the various magazines, I found out there were a few other shells for sale, and decided to risk $50 on *Tshell*. This was the best software investment I had yet made. I used this shell for a couple of months and then ordered *Wshell,* priced $50 also, to see what made it different. There aren't as many articles on shells as there are on games, so I had to risk some money to see if it was useful. It was.

Using any of these shells made working with either the Manx or Lattice compilers easier, and I couldn't begin to add up all the time saved by using them instead of the plain CLI. I use four different shells for different reasons; they all have their strong points. This article will discuss the features of these shells, so you can make a decision as to which one to use. I cannot understand why anyone would want to use the CLI without a shell; if you give them a try, I don't think you'll be disappointed. In this article, I'll talk about Dillon's shell, Tshell, and Wshell, as well as the AmigaDOS 1.3 shell.

The AmigaDOS shell

The AmigaDOS 1.3 shell version 1 comes with the Workbench 1.3 update at no extra cost. This shell is the easiest of the four to get started with: just double-click on the shell icon from Workbench, and it is up and running. The window opened is resizable, and its default size, position, and title can be changed via a 'tooltype' in the icon.

The 1.3 shell also has command line history, a feature provided up to now by Bill Hawes' shareware *conman* program. The 1.3 command history uses a 2k buffer, and it enables you to use the up and down-arrow cursor keys to scroll through the previously-issued commands and re-enter any one of them. Command history is one of the greatest benefits of any shell (they all have this feature in one form or another), and it can save a lot of typing, especially in repetitive work like compile directives with lots of options. The AmigaDOS shell also has key-sequences to search for a specific command in the history, and to move to the bottom of the history list.

Unlike the simple CLI, the 1.3 shell has command line editing, which lets you cursor back and forth through the current command line and edit it before pressing RETURN. You can also move the cursor instantly to the beginning or end of the command line, delete to end of line, delete the entire line, or tab the cursor.

Aliases are supported by the 1.3 shell, which is a must for any serious shell. Aliases let you define your own mini-commands, and are often used to insert certain options within a standard system command. For instance, if you define an alias like this:

```
alias adir "dir [] opt a"
```

you may then enter 'adir df1:' instead of 'dir df1: opt a'. Aliases can be global (automatically set up every time you run a shell) by listing them in the 'shell-startup' file in the 's:' directory, or local, by entering them at an opened shell. Additionally, there are two script files in the 's:' directory called SPAT and DPAT, which can give pattern matching to aliased commands that normally lack that option. (S is for single argument, and D is for double argument pattern commands.) There is also a 'PCD' script file that remembers your current directory, letting you temporarily visit a new directory and easily return again.

The 1.3 shell also uses the redirection arrows <, >, and >>. These are used to change where the input and output goes, like to the printer instead of the screen, using the filename '>prt:'.

Another useful feature is the shell's support of the script file protection bit, which lets you execute script files without having to type 'execute'. You can put useful DOS scripts in the 's:' directory, use the *protect* command to set the script bit, and from then on use them as if they were normal CLI commands.

Probably the best feature of the AmigaDOS shell is that it comes with the 1.3 update, and anyone can use it as a CLI substitute. Those who do will quickly see the advantages of using a shell over the CLI. I prefer the other shells over the AmigaDOS shell; they seem to be aimed more directly at programmers than this one, which lacks some important features.

The Matt Dillon shell

Matt Dillon's shell is in the public domain, and is on the Fred Fish disks (called 'csh') and also comes with the Lattice C 4.0 Compiler. Source code is included. This shell is constantly updated by Matt and others in the Amiga community, and the current version is 2.10. The constant updates are one of the best reasons to use Dillon's shell. There are a couple of versions modified by other programmers, and they also deserve a look. Matt has provided a tremendous amount of valuable software for the public domain: just look at Fred Fish disks 168 and 169, which contain the source code and executables for just some of his software.

Dillon's shell supports script files that are more powerful and efficient than AmigaDOS scripts. Its aliases are much more powerful as well, since you can use variables, multiple arguments, conditional expressions, and even string operators - it's actually a miniature programming language. The shell also supports I/O redirection, and piping, which lets you direct the output from one command into the input of another.

Dillon's shell is easily a commercial grade program, and it is full of options to please any programmer or user. This shell is a must-have for all Amiga owners - yes, even the Workbench-only types. The Help key is active, and it gives a list of all built-in commands. Speaking of which, there are over 30 built-in commands that are a part of the shell itself, and so do not need to be loaded in for use. These commands do not all have the same names as their AmigaDOS equivalents, (like 'mkdir' instead of 'makedir'), but they perform the same way. The commands can also be abbreviated to one or two characters to save typing. This shell is simple to install on a disk, and as with all the other Dillon public domain programs out there, all you need to get started is Fred Fish disks 168 and 169 for the shell and tools. Matt is a programmer, and has included much of what he feels useful in his shell.

Tshell

Tshell is written by Jay Ts and is published by Metran Technologies, with 1.0g the most current version. Tshell is a commercial program that runs about $50, and is worth every cent of it. It uses the Help key to bring up the list of commands available from the complete on-line manual, like Dillon's shell, but it also explains how to use any of the commands or tools that come with it. These explanations cover definitions, usage, options, and detailed examples! The documentation is disk-based, and can be copied to the storage device of your choice (preferably RAM, if you can spare the memory). The response from df0: is slow, but is by no means unbearable. The documentation takes up quite a bit of disk space, but you don't need to use it if you can't spare the room. Single-drive users will have a dificult time deciding what to remove from the main disk if they want to set it up with additional utilities.

This shell is both easy and fun to use, and those that have stayed away from the CLI before now owe it to themselves to look at this program. I really have fun working with AmigaDOS when I'm using Tshell.

Like Dillon's shell and Wshell, Tshell has built-in commands, and there are over 40 of them. These commands are actually on disk, but can be loaded and used as built-ins. Some of the commands are not compatible with their AmigaDOS counterparts, so you should keep the AmigaDOS or ARP commands around if you want all the standard options.

Load will allow loading into memory either built-in commands, the system segment for aliases and I/O redirection, or both. I have two megabytes of memory, and so can copy in all the built-in commands *and* the on-line manual, but then there isn't enough room for my compilers, so I must use a disk-based compiler setup.

Load is also used to load in commands as memory resident. The options for residency are AmigaDOS-type residency or RAMdisk residency. AmigaDOS residency is similar to that of AmigaDOS 1.3, but there is an additional option called Protect that insures that these commands won't be multiply executed. RAMdisk residency is a safer way of making any command resident, although it is slower than AmigaDOS residency.

Unlike the other shells, Tshell is written as a package with all necessary tools included. It has the usual command line history and editing features, and also supports the 1.3 AmigaDOS protection bits. Unlike the other three shells, this one cannot be started from Workbench.

Tshell has an implicit *cd* command, which means that you can simply type the destination directory's name and do not have to precede it by 'cd'. Aliases are also supported in Tshell, using the symbols ':=' for defining them instead of an *alias* command.

The window's title bar displays the CLI process number, current directory, and the value returned by the last command. It's nice not to have to depend on another program to provide memory usage and the time. The clock displays hours, minutes, and seconds and is updated constantly. The title bar display used to disappear after a while in older versions of Tshell, although it appears that bug is removed in the current version. ARexx is supported; this will hopefully be an important point for future software.

At 80k, Tshell is the largest of the shells. This can be a drawback if you like opening up several shells at once, although that is not reason to reject it, as the features outweigh the size penalty. The author provides his own customer support, and

has never failed to help me out. He is currently working on a companion disk with more tools for Tshell.

Tshell actually provides a UNIX-like interface to the CLI, and many of its commands will sound familiar to C programmers: such as printf, strlen, continue, break, exit, and return. It even has a C-like environment where simple C programs can be written, and they don't need to be compiled to run - as the shell acts as a C interpreter! This feature makes Tshell stand out from the other shells. There is a directory with some programs written for Tshell to run, and the programs are similar to examples found in C programming Books.

Wshell

Wshell is the best-written program I have used for the Amiga yet. It is written by Bill Hawes, author of ARexx, and is the premier programmer's shell for the Amiga. This is the only Amiga shell written completely in assembly language; Dillon's shell and Tshell are both written in C. Speaking of ARexx, Wshell has a built-in ARexx interface, which means these two programs go together quite well, but they can can also stand alone. There are a number of extra options, when compared to the AmigaDOS 1.3 shell, that are time-savers and highly useful for us end users. The manual is well written and organized, and is spiral bound, so it stays flat and is easy to refer to while working with the shell. The most version is 1.05, and is the first update to the original release.

Wshell is written without command line history/editing as a feature of the shell itself; you use *conman* instead - it also was written by Bill. A copy of conman is supplied on the disk with Wshell, and there is a chapter devoted to conman and its options.

There are twenty commands built into the shell itself; they add little to the size of the code, according to Bill. Most commands are replacements for common AmigaDOS ones, with some additions. *Pushcd* is used to save the current directory so that it can be returned to by using the *popcd* command. There can be more than one directory stored with *pushcd;* they are stored last-in-first-out. Each use of *popcd* moves to the next directory in line.

Swapcd is used to exchange the current directory and the first directory in the list stored by *pushcd.* Although *cd* is used to change directories, just like the AmigaDOS *cd* command, Wshell - like Tshell - has an implied cd.

The *if* and *echo* commands were changed for compatibility with 1.3 AmigaDOS. *Goto* is not used in AmigaDOS, however, it is used in Wshell to redirect flow within a script file.

The text displayed in the title bar and prompt can contain very useful information. Wshell has a considerable number of codes that can be in these strings to let the user select what kind of information he wants displayed. For instance, my prompt defined as: ' %T Elapsed=%E %Y'. This will display the current time, time elapsed since the last press of the RETURN key, and error code (0 if no errors). These values are updated each time the return key is pressed; the events are not constantly changing values, although another option will allow that. My title bar string displays different information - I want to know how much memory is free each time I open a new window, so I set it as: [CHIP %MC] [FAST %MF] [PUBLIC %MP]. You can display a variety of information in the title bar, like the current directory, elapsed time, failure level, task ID, chip, fast, or public memory free, task number, task priority, return code, stack size, time, and error code.

Like the other shells, Wshell supports piping, using PIP: from *conman.* Wshell also supports the script bit: use AmigaDOS 1.3 *protect* to set the bit, and Wshell detects it, even without using the other 1.3 commands. The shell sees the script bit and adds *execute* to the start of the command, so you should make *execute* resident to speed things up. Wshell also comes with a program called *scriptify,* which sets the script bit for those that haven't got around to buying AmigaDOS 1.3 yet. The *alias* command is supported, using '[]' to specify arguments within the string the same way that AmigaDOS 1.3 does.

Some additional comments concerning Wshell should include the ease of use and shared-library features. This shell is quite simple, although it is packed with power. There is no need to have most of the options on to use the shell effectively. Wshell is extremely reliable - I have not had a crash caused by it in the year I have spent working with it. It uses shared libraries, which means that it only takes 10k of memory for each additional shell opened! As for customer support, Bill lists his phone number in the manual, and has always taken the time to answer any of my questions. I wish all programmers were this accessible; although some of them are, most are not at the publisher's where they will spend time to talk to you personally. Good customer support is a must for any program, and Bill gives great customer support!

Worth shelling out for?

A shell is a must-have for anyone doing programming, or just manipulating files on disk using the plain CLI. The price at $50 is more than reasonable - just forego buying one of those nice games this week, and get a tool that you will use constantly. Both Tshell and Wshell updates are quite reasonable: Wshell was $5, and the Tshell update to version 1.1 is free when you return your original disk.

A shell is not something that you will get tired of in a couple of weeks, and you'll use it constantly. You are not just exploring a program, but the Amiga itself when using the AmigaDOS commands and a shell. While not as simple as icons, there is a lot more that can be done from the shell in terms of flexibility and speed. It doesn't take a programmer or a genius to use a shell - anyone that uses an Amiga should have little or no trouble at all. Considering how many people have already learned how to use the CLI, there are plenty of users out there to go to if you need help.

MIDI Programming Primer

Key concepts in MIDI protocol

by Daniel Brookshier
Copyright © 1989 Daniel Brookshier

Why would you want to write your own MIDI software? In my case, I have trouble humming a tune, let alone trying to play Beethoven or Pink Floyd on my Casio keyboard. My interest lies more in the *Instant Music* types of software. I dream of software that prevents the novice from playing music that sounds like the torturing of furry animals. But I find very little time to attain such a lofty goal. So I will let you in on the secrets that I have learned about MIDI and the Amiga, in hopes that you will save the world from notes hit out of key.

What is MIDI? To start with, MIDI stands for Musical Instrument Digital Interface. It allows electronic musical instruments and computers to interface and to control each other using a standard communication language. Using MIDI, a composer can record music directly from the synthesizer keyboard. Once a section of music has been recorded, it can be replayed or modified. During a live performance, sets of instrument characteristics can be modified by the press of a button. This reduces the burden on the musician, who may need to change dozens of instrument parameters while playing.

On a basic MIDI instrument there are three five-pin DIN plugs labeled 'In', 'Out' and 'Thru'. The 'In' is connected to the 'Out' of another instrument or computer sending messages to the instrument. The 'Out' connector is used when messages from the instrument or computer are to be sent to other instruments or computers. The 'Thru' connector is just a pass through of the signals coming from the 'In' connector. The 'Thru' connection allows several instruments to be controlled by one source.

Midi.mod is a Modula-2 module containing all the procedures needed to open, close, read and write to/from a MIDI device. (***Midi.mod*** *and related modules can be found on the disk for this issue. Due to space restrictions, they are not reproduced here. – Ed.*) Using MIDI is very similar to normal communications done with a modem. In fact the setup for MIDI is almost the same for a modem transmitting eight bits of data with one stop bit and no parity. The only differences are in the ioSerFlags of the serial device and the special baud rate. The flags SerRadBoogie and SerXDisabled must both be set and the baud should be at 31.25K for MIDI. Once the serial device has been initialized, reading or writing MIDI codes is accomplished by sending messages to the device port.

The signals that MIDI instruments send and receive come in five formats. They all consist of at least one byte; most messages are two or three bytes in length. The first byte is the *status* byte, which is sent to signal a new event. It has the first (most significant) bit set. This can be checked by verifying that the value of the byte is greater than 126.

The status byte can be followed by one or more *data* bytes. Data bytes have the first bit cleared. This means that we can tell if it is a data byte by checking if the bit is zero or by verifying that the value of the entire byte is less than or equal to 126.

The best way to determine what your going to do when you receive a byte from the MIDI channel is to use the following structure:

```
IF MIDI_In > 126 THEN
   (* Handle status byte *)
ELSE
   (* Process data byte *)
END;
```

We must next decode the status byte. There are five types of status byte, each handling a different level of control. The five types cover two groups of MIDI messages: *channel* messages and *system* messages. In Figure 1, I have given the bit patterns of these MIDI messages along with the formats for the data bytes.

To start the decoding, the specific message group must first be

determined. The status byte is first broken into two sections, an upper and lower half. The type and group of the command is in the upper, most significant half. The lower half contains either the channel number or the subset of the command. Decoding these numbers can be done in several ways. The first task is to mask out the status and channel separately into two new bytes. In C this can be quite simple. As shown below, we just bit-wise AND a constant and the status byte.

```
Command = 0xF0 & MIDI_In;
Channel = 0x0F & MIDI_In;
```

For Modula 2, a different approach is needed. We must first transform the status byte into a set. Next the intersection of the mask and status sets are determined. Then the result is converted back to an unsigned byte. In the example below, the values to be ANDed together must be converted to sets in order for the compiler to interpret the * as an AND instruction. Next the value is converted to a CARDINAL type (unsigned integer).

The operation is the same as in C except the terminology has been changed. Also the upper half of the word is shifted to the right so that the number only ranges between 8 and 15. This makes the code a little simpler to read later on.

```
TYPE
   MaskSet : SET OF [0..7];
VAR
   MIDI_In : BYTE;
   Command,Channel : CARDINAL;
BEGIN
   .
   .
   .
CommandMSB = CARDINAL(SHIFT(MIDI_In,-4)) ;
CommandLSB = CARDINAL(MaskSet(00FH) *
    MaskSet(MIDI_In)) ;
```

Now that the command has been broken into nibble-sized chunks the process of determining the message group and the command is straightforward. The procedure DecodeMIDI-Command in *readmidi.mod* does most of the work. The procedure extracts the data from the status byte, then uses CASE statements to determine the type of commands. As can be seen in the upper half of the status byte, the channel messages range between 8 and 14 while the system messages are 15. For channel messages, the lower half of the status byte contains the channel number. For system messages, the lower half of the status byte contains the specific type of system command.

There are four different actions that are taken, depending on the type and the group of the command. If it is not one of the standard MIDI commands, a procedure is called to notify the user that an error has occurred. If the command is one that will be followed by one or more data bytes, the number of bytes to wait for is set and the data byte counter is reset. Also the channel number and the command are saved, to be used when all the data has been received. If a command has been received that needs no data, the action needed by the command is performed immediately by calling another procedure.

Finally, if the program does not support the command, then an alternative procedure is called to inform the user that that MIDI feature is not supported. Caution should be used in printing such messages: a series of unimplemented or erroneous commands would cause the messages to be printed out faster than they could be read!

For both the erroneous and unsupported commands, a variable is set to allow the program to ignore any data that may be sent after it.

Once the status byte is decoded, the following protocol rules must be followed:

Rule 1: Each status byte, if it needs data, has a specific number of data bytes expected in the following data packet(s). No action should be performed until a complete data packet is received.

Rule 2: After a status byte has been received, any number of complete data packets may be received before another status byte arrives.

Rule 3: System messages can be received at any time, even between data bytes!

Rule 2 is the most important. If a key is pressed, for example, the keyboard will send a status byte and two data bytes. If two keys are pressed at about the same time, then the MIDI device will send one status byte and two sets of two data bytes.

Rule 2 also causes problems with a variation in the Note On command. On some keyboards the Note Off command is not used. Instead, these keyboards use a velocity of zero in the Note On command to signify that a note should be turned off. MIDI programs must take this into account to prevent notes from being stuck in the on position.

Rule 2 also means that if a key has been pressed before your program has been started, the program will have never seen the status byte. The program will be getting data that it does not know what to do with. The MIDI program should always ignore data until a status byte is received.

I have now reached the point where the commands can be explained. I will only go into those that most keyboards use. I am at a disadvantage, because the keyboard that I own has very few functions that are used with MIDI. As an aside, I should warn the prospective buyer of a MIDI controlled instrument to check the manual for supported functions. The Casio HT-700, for example, does not provide MIDI support for ninety percent of its features. This is because the ability to customize the sounds on the instrument is only possible through a wheel type device. Just to change the characteristics of one instrument could take several minutes.

```
   Channel Voice Messages
Note Off              1000nnnn  0kkkkkkk  0vvvvvvv
Note On               1001nnnn  0kkkkkkk  0vvvvvvv
Polyphonic
   Key Pressure       1010nnnn  0kkkkkkk  0ppppppp
Control Change        1011nnnn  0ccccccc  0sssssss
Program Change        1100nnnn  0iiiiiii
Channel Pressure      1101nnnn  0ppppppp
Pitch Bend Change     1110nnnn  0bbbbbbb  0BBBBBBB


   Channel Mode Messages
Local Control         1011nnnn  01111010  0LLLLLLL
All Notes Off         1011nnnn  01111011  00000000
Omni Mode Off         1011nnnn  01111100  00000000
Omni Mode On          1011nnnn  01111101  00000000
Mono Mode On          1011nnnn  01111110  0000mmmm
Poly Mode On          1011nnnn  01111111  00000000


   System Common Messages
Song Position Ptr     11110010  0SSSSSSS  0SSSSSSS
Song Select           11110011  0ttttttt
Tune Request          11110110


   System Real Time Messages
Timing Clock          11111000
Start                 11111010
Continue              11111011
Stop                  11111100
Active Sensing        11111110
System Reset          11111111


   System Exclusive
System Exclusive      11110000  0aaaaaaa  0zzzzzzz...
End Exclusive         11110111


   Key to bit patterns
nnnn       Channel number.
kkkkkkk    Pitch in half steps, with middle C = 60.
vvvvvvv    Key velocity.
ppppppp    Key pressure.
iiiiiii    Patch or rhythm number.
ccccccc    Controller parameter device number.
sssssss    Controller parameter value(see Figure 2).
bbbbbbb    Least significant byte of pitch bend.
BBBBBBB    Most significant byte of pitch bend.
LLLLLLL    Local control, on = 127, off = 0.
mmmm       Maximum number of channels to receive.
xxxxxxx    Least significant byte of song position.
XXXXXXX    Most significant byte of song position.
tttttttt   Song selection number.
aaaaaaa    Company ID.
zzzzzzz    Exclusive data.
SSSSSSS    Song event position number.
```

Figure 1.

The first and most important command for those looking into high performance instruments is the *system exclusive*. The command starts with a status byte like the rest. Next comes a data byte containing the company identification. Following this byte is a series of bytes containing special data for the particular instrument. The complete command is then terminated with an end of system exclusive status byte. Although there are some standard formats that are common between companies, you should refer to your manual for further information.

The next set of commands are the channel messages. These messages have the channel identification number that is decoded from the lower half of the status byte. The number ranges between 0 and 15, meaning that up to 16 MIDI instruments can be controlled on one MIDI cable.

The channel number shows what instrument or what part of the instrument we are receiving the data from or sending data to. The channel number should be processed because we may receive a channel message from an instrument that we want to ignore. Remember that MIDI allows several instruments to be connected on the same cable! On the Casio HT-700 keyboard, channel numbers are also different for the rhythm and the main keyboard.

The first of the channel messages deal with the notes to play and how they are controlled. Note On and Note Off signify which specific note to play or stop playing. The data bytes of both commands have the note and the velocity of an instrument being played. The note is determined by starting with the value of 60 equal to middle C. Adding one or subtracting one to this number adds or subtracts one half step. On a piano, a half step is the distance between adjacent white and black keys, between the B and C keys, and the E and F keys. The values for notes centred on middle C would have the following note values:

49 = C#	56 = G#	63 = D#
50 = D	57 = A	64 = E
51 = D#	58 = A#	65 = F
52 = E	59 = B	66 = F#
53 = F	60 = C	67 = G
54 = F#	61 = C#	68 = G#
55 = G	62 = D	69 = A

The MIDI key value must be converted to frequency to be used by the Amiga audio device. The procedure `MIDI2FreqConv` is called to create a table of frequencies. The MIDI key number is used as an index into the table. The equation used is quite simple: it multiplies a constant by the last entry in the table. The output is quite accurate.

The velocity of the note is the force with which that the note is played. An example of this would be force used when striking a key on a piano. If the key is hit with a light touch, the sound produced is low volume. If the key is struck harder, the sound produced is louder. The amount of extra electronics needed to make a keyboard record velocity is expensive: keyboards are

usually just ON/OFF switches. The MIDI devices that will always use this data are the drum machines. The velocity represents how hard the drum has been hit.

The Note Off command is similar to Note On. The only difference is that the velocity determines how fast the sound decays to silence. Please remember that this command may be substituted with a Note On velocity value of zero.

Polyphonic Key Pressure is another command that may be available only the more expensive keyboards. This feature represents how much constant pressure is being used to play a specified note. Most explanations of this command say that it is used to signify vibrato. It may also be used in the newer wind instruments, like the new MIDI saxophone. The pressure value would show how hard the player is blowing into the instrument.

Program Change is what is usually called 'patch number'. On an instrument with several pre-programmed instruments, Program Change switches the selected instrument. On some keyboards, Program Change can be used to select the rhythm.

Control Change is a versatile command allowing control of several different types of parameters. Figure 2 is a list of some of the possible values that may be modified. The selection of the parameter is in the first data byte and the new value for the parameter is in the second byte. Remember that not all of these may be recognized by your instrument.

Figure 2: Controller Parameter Device Numbers

1. Modulation wheel	67. Soft pedal
2. Breath controller	69. Hold 2
4. Foot controller	80-83. General controllers
5. Portamento time	92. Tremolo
6. Data entry MSB	93. Chorus depth
7. Main volume	94. Celeste
8. Balance	95. Phaser depth
10. Pan	96. Data increment
11. Expression controller	97. Data increment
16-19 General controllers	98. Un-registered parameter LSB
32-63 Data entry LSB	99. Un-registered parameter MSB
64. Sustain pedal	100. Registered parameter LSB
65. Portamento	101. Registered parameter MSB
66. Sostenuto	122. Reserved for mode messages

**Unlisted numbers are not defined in MIDI 1.0 specification.*

Channel Pressure is similar to polyphonic pressure except that it applies to everything on the MIDI channel. Any notes currently playing would be affected.

The last channel message is Pitch Bend. This is one of the more fun commands, and is also on most new keyboards. The pitch bend wheel data starts at 8192, which means that 8192 is subtracted from the pitch bend value and then added to the value of all the notes currently playing. For HT-700 and other keyboards, only the MSB is used, so the data starts at 64. In my example, each pitch is added directly to the period sent to the Amiga audio device.

The *channel mode* message is used to control the keyboard characteristics. The Local Control Off command allows the keyboard to be used only as an output device without playing any of its own instruments. This is useful when the keyboard is only used as input to the Amiga or other MIDI devices.

The next channel mode message is the All Notes Off command. This does exactly what it says: it turns off all the notes. It is a good command to have if some of the notes get stuck on.

The rest of the channel mode messages have to do with how many notes can be played at once. The Poly commands allow two or more notes to be played at once. The Omni mode only allows one note to be played at a time. If more notes are sent than are expected, the newer notes will be ignored.

The next set of MIDI commands are the system *real time* messages, which only consist of one byte. It is important to remember that these may appear between other messages. If you have been receiving note events and then get a system real time message, then the next data byte should still be handled as a note event.

These messages are all associated with sequencing. They control the starting, stopping and timing of events. The most important of these commands is Timing Clock. This command is sent every time the lights blink on the rhythm section of most keyboards, thus acting as a metronome. It is best used as a constant update for timing. Because not all instruments are alike, a common timing source is needed. An instrument with a rhythm section should have a switch to toggle between sending or receiving this command.

The main module `ReadAllMIDI` in *readmidi.mod* reads the basic MIDI note and pitch bend commands and plays up to four channels of a triangle wave through the audio device. As each command is received, the note value is converted to frequency and the octave is determined. If a channel is opened, the sound is started on that channel. If the pitch wheel is moved, the new value is added to the wavelength to play. When the MIDI instrument key is released, the sound on the corresponding channel is stopped.

If new MIDI channel commands are to be decoded, only the command value and code need to be added to the `CASE` statement.

To round things up, MIDI is a very capable interface that allows a lot of flexibility. The program example on this issue's disk falls far short of using the complete set of commands possible, but they can be added with very little work. Just remember the rules and don't hit any notes out of key.

Checking Stack Usage

How much stack is your program using?

by Nick Sullivan

When you write a program for the Amiga, one of the many things you have to take into account is the amount of stack space your code will require. If you happen to be writing in a high-level language such as C, it can be quite difficult to come up with an accurate estimate of your stack requirements other than by trial and error. This is especially true if your program employs recursion, or if you are addicted to stack-hungry practices like allocating large 'auto' arrays.

The little assembler module below provides a simple and convenient method of determining stack usage from within a running program. Its method of operation is straightforward. Near the beginning of your program, you make a call to the routine *_PrepStack*, either with:

```
jsr _PrepStack
```

if you're writing in assembler; or in C, simply:

```
PrepStack();
```

This will cause a portion of your stack to be filled with the value $71e9 (the variable CODE), an arbitarily chosen number you are free to change in the source if you want. The amount of stack filled is set (in *words,* not bytes) by the variable DEPTH. You'll want to change this value to suit your application, since DEPTH *must* be greater than the number of words of stack your program will use. Of course, you don't yet know how much that is, or you wouldn't be doing this at all, so make it something big and safe. You must also remember to set your stack size (with the CLI *Stack* command) to an even bigger value, to allow for stack used by your program's start-up code. If you set DEPTH to 10,000, for instance (20,000 bytes), it might be appropriate to do a **STACK 25000** before running the program. After the call to *PrepStack()*, your program should continue as usual, doing whatever it is that it is supposed to do. Just before it exits, and from the same function that called *PrepStack()*, you should call the routine *_CheckStack*; either:

```
jsr _CheckStack      ; result returned in D0
```

or:

```
count = CheckStack();  /* count is a long int */
```

_CheckStack examines the stack space it previously filled with the CODE value, starting at the end with the lowest memory address and working up towards the stack pointer. When it first finds a value other than CODE, it subtracts its current working address from the stack pointer and returns the resulting value, the number of stack bytes you have consumed. Bear in mind that the amount of stack a program uses may vary from run to run depending on what inputs are supplied. In a recursive program, the amount of variation may be considerable, so always try to provoke worst-case conditions in your test runs.

```
* _PrepStack/_CheckStack
*
* by Nick Sullivan
* Kwik Filler Enterprises, March 22, 1989
*
* Stack-checking routines callable from C or assembler.
*
* Usage:
*
*       long count, CheckStack();
*
*          PrepStack();          /* near front of program */
*          /* your program's main business here */
*          count = CheckStack(); /* near end in same function */
*          printf("Stack usage: %ld\n", count);

        xref    _PrepStack
        xref    _CheckStack

DEPTH   EQU     1500    ;max stack check size in words == 3000 bytes
CODE    EQU     $71e9   ;arbitrary code word to fill stack with

_PrepStack:
        move.w  #CODE,d0                ;code word
        move.w  #DEPTH-1,d1             ;set up loop count
        move.l  sp,a0                   ;current stack pointer
ps_wl
        move.w  d0,-(a0)                ;fill stack with code word
        dbf     d1,ps_wl

        rts

_CheckStack:
        lea     -DEPTH*2(SP),a0
cs_wl
        cmpi.w  #CODE,(a0)+
        beq.s   cs_wl

        move.l  sp,d0
        addq.l  #2,d0                   ;size of return address-2
        sub.l   a0,d0
        rts

        end
```

Process Spawning

Multitask from within your programs

by Chris Zamara

In many cases, multitasking from within a program can be an excellent programming solution. For example, a text editor that supports multiple documents could simply create a new task for each document, using the main editor routine each time. This approach not only simplifies the design of the software, it is also advantageous to the user: since each document is managed by a separate task, it can be edited regardless of what operations are going on in other documents. The same principle can be successfully applied to all kinds of programs.

Unfortunately, turning routines in your program into separate tasks isn't totally straightforward, as you may have discovered if you've attempted it using Exec's AddTask() function. The problem is that in order to perform AmigaDOS-based I/O of any kind from within the routine, it must be a *Process*, not just a task. This is because a *Task* is handled by Exec, the part of the system that controls multitasking, but unless a *Process* is created, AmigaDOS knows nothing about the task and can't support it in any way.

The solution seems to be the DOS function CreateProc(), but as soon as we start dealing with AmigaDOS, things get a little bizarre. For one thing, CreateProc() expects as one of its arguments a pointer (actually, a *BPTR*) to a *segment*. We can get this by calling LoadSeg(), but that will load a program from disk; we want to use an existing routine (which has already been loaded with the rest of the program) as the process.

The simple solution is to use the program listed here, and call the spawn() function whenever you need to spawn a new process. You use it like this:

```
success = spawn(function, name, argc, argv);
```

...where 'function' is a pointer to the function you want to spawn as a process, 'name' is a process name (like "MyProc"), and 'argc' and 'argv' are command-line-style arguments that get passed to the routine. You must declare these arguments in the function you are spawning (e.g. int argc; char **argv).

Just compile 'spawn.c', link with 'spawn.o', and call spawn() from within your program to create a new process at any time. A nonzero return value indicates a successful process launch.

The global variable 'ProcRunCount' holds the number of spawned processes currently running. Make sure that your program doesn't exit until this is zero, otherwise you will unload the code used by the running processes and the Guru will undoubtedly make an appearance. Also, if you are going to spawn a given routine more than once, the code must be completely reentrant. That means *no* global variables or static data of any kind may be modified by the routine.

This code requires Manx Aztec C, since it uses the '#asm' directive, and the geta4() function. If you have Lattice C version 5.0, you should be able to make it work by assembling the assembler portion separately.

The program works using a trick I found in a device handler written by Matt Dillon: in order to create a fake segment whose address can be converted to a BPTR, Matt just used a bit of assembler with a few NOP instructions before a call to the C routine. Added to this is a simple message-passing protocol that passes arguments to the spawned process, and simplifies file access by giving the new process the same 'current directory' as the main one.

```
/* "spawn.c"

   This lets you spawn a new process from code that is part of your main
   program, i.e. without having to load from disk. Note that if you plan to
   spawn a routine multiple times, the code must be reentrant!

*/
#include <intuition/intuition.h>
#include <functions.h>
#include <libraries/dos.h>

#define STACKSIZE (4000)   /* stack size for new process */

int ProcRunCount = 0;      /* how many are running at any time */

struct ProcStartMsg {
   struct Message Msg;
   void (*func)();         /* ptr to function to spawn as process   */
   short NumArgs;          /* passed to spawned function            */
   char **Args;            /* args in argv[] style, also passed     */
   struct FileLock *lock;  /* lock of cd for CurrentDir()           */
};

void start_proc();

struct MsgPort *ProcSend, *ProcRply;

#define CTOB(x) (void *)(((long)(x))>>2)  /* convert pointer to BPTR */
```

```
spawn (func, procname, numargs, args)
void (*func)();
char *procname;
int numargs;
char *args[];
{
struct ProcStartMsg startmsg;

   if ((ProcSend = CreatePort(NULL, NULL)) == NULL)
      return 0;
   if ((ProcRply = CreatePort(NULL, NULL)) == NULL)
   {
      DeletePort(ProcSend);
      return 0;
   }
   startmsg.NumArgs = numargs;  /* parameters sent to new process */
   startmsg.Args = args;
   startmsg.func = func;

   startmsg.Msg.mn_ReplyPort = ProcRply;
   startmsg.lock = Lock("", ACCESS_READ); /* find lock of cd */

   /* put startup message and start process */
   PutMsg(ProcSend, &startmsg);
   if (!CreateProc(procname, 0L, CTOB(start_proc), (long)STACKSIZE))
      return 0;
   WaitPort(ProcRply);
   /* got reply, process is happy. */
   GetMsg(ProcRply);
   DeletePort(ProcSend);
   DeletePort(ProcRply);

   return 1;
}


do_procmain()
{
struct ProcStartMsg *msg;
void (*func)(); /* function we'll call */
int argc;
char **argv;
struct FileLock *cdlock;

   geta4(); /* fix a4, since we don't enter through normal startup code */

   WaitPort(ProcSend);     /* wait for start message    */
   msg = (struct ProcStartMsg *)GetMsg(ProcSend);

   argc = msg->NumArgs;    /* pass these to new process */
   argv = msg->Args;
   func = msg->func;
   CurrentDir(cdlock = msg->lock); /* set cd            */

   ReplyMsg(msg);          /* reply to start message    */

   ProcRunCount++;
   (*func)(argc, argv);    /* call new process's main() */
   ProcRunCount--;

   UnLock(cdlock);
}


#asm
   public   _start_proc
   public   _do_procmain

   cseg
_start_proc:
   nop
   nop
   nop
   nop
   nop
   movem.l D2-D7/A2-A6,-(sp)
   jsr   _do_procmain
   movem.l (sp)+,D2-D7/A2-A6
   rts
#endasm
```


SAFE HARBOR SOFTWARE & PERIPHERALS

Great Prices! Shipping based on weight and zone

For Information & Catalog Call
Voice: 414-544-2066
Pursuitable BBS: 414-544-6567

Software

ARexx	31.10
B.A.D.	33.00
CLIMate	27.73
Comic Setter	60.00
Copyist	159.10
Cygnus Ed	64.69
Devpac Amiga	64.69
Digi-View Gold	143.10
Distant Armies	28.30
GFA BASIC	88.00
HISoft BASIC	150.00
Hostage Rescue Mission	30.81
KCS 1.6A	145.00
Lattice C 5.0	190.00
Lords/Rising Sun	33.89
Masterpiece Fonts	155.00
Micro Fiche Filer +	111.00
Micro Lawyer	37.00
Movie Setter	60.00
PageStream	131.00
PIXMate	42.30
Police Quest	33.89
Professional Draw	130.00
Project D	32.00
Quarterback	46.21
Raw Copy	39.37
Superbase Prof 3.0	200.00
Super Plan	99.97
Turbo Silver	120.00
Works! Platinum	171.00

Books/Diskettes

Amiga 3D Graphics	16.30
Advanced Amiga BASIC	15.00
Amiga C for Beginners	15.00
Amiga Machine Language	16.30
Amiga Systems Programmer	26.25
ROM Kernal Referance	27.50
DS/DD Bulk Sony 50 pk	73.00

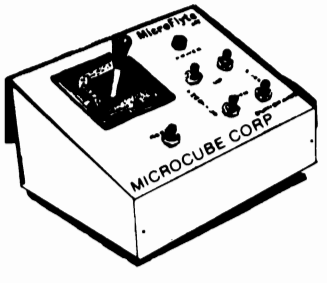
Hardware

AMAS Stereo Digitizer	128.00
Amigen	145.00
Auto Droid	50.00
Color Splitter	80.00
Floppy Internal 2000	115.00
Floppy, Unidrive	136.25
Floppy, Twindrive	268.00
Hardcard 40Q 2000	798.00
Hardcard 80Q 2000	1135.00
Harddrive 40Q SCSI	580.00
Harddrive 80Q SCSI	915.00
Live! 2000	315.00
Live! 500	298.00
Memory Board 501 Clone	180.00
MIDI Gold (2000)	67.00
MIDI Star	207.95
MiniMegs 1000 512K	330.00
Modem, Supra	140.00
Mouse, Boing!	110.00
Perfect Sound	66.93
Perfect Vision	185.00
Spirit Brd 500/1000 0K	212.56
Spirit Brd 500/1000 512K	391.76
Supra SCSI Cntrlr 2000	205.00
Supra SCSI Cntrlr 500	176.60
Tool Box 500/1000 3 amp	215.00

Orders Only Please:
Outside WI: **800-544-6599**
Inside WI: **414-544-6599**
Visa/MC/CODs

2414 Pendleton Place ■ Waukesha, WI 53188 ■ 9 AM to 5 PM M-F

NOW AVAILABLE FOR THE AMIGA



The MicroFlyte JOYSTICK, the only fully proportional continuously variable joystick control for Flight Simulator II

". . . It transforms an excellent program into a truly realistic flight simulation system" B.A.C.E.

MICROCUBE PRODUCTS

Commodore 64/128
- MicroFlyte ATC Joystick **$59.95**
- Test/Calibration Disk:
  A diagnostic tool for your joystick **$4.95**

Amiga
- MicroFlyte Joystick—Plugs into the mouse port & works with most software **$119.95**
- Analog Joystick **$74.95**

Include $4.00 shipping of joystick orders. FSII is a trademark of subLOGIC Corp.

Order Direct from:

MICROCUBE CORPORATION
P.O. Box 488
Leesburg, VA 22075
(703) 777-7157

Using the Bison/Yacc Parser Generator Part 2

Creating 'ECalc', an expression calculator

by Eric Giguère
Copyright © 1989 Eric Giguère

Eric Giguère is a computer science student at the University of Waterloo, and has been involved in the design of an ANSI-conformant C compiler with the Computer Systems Group.

Last issue we took a look at the theory behind using a parser generator in the creation of a compiler or interpreter. In this final instalment we put some of that theory into practice by using the Bison parser generator to build a small but powerful expression calculator. (Strictly speaking, the calculator is an interpreter rather than a compiler. I considered several projects that would generate code of some kind, but all were too long for even minimal publication. As it is, only part of the source code for the calculator is published in the magazine due to space limitations. The complete source code is available on disk, however, as well as a copy of the Bison executable. Source code for Bison is available on Fish Disk #136.)

Using Bison

Bison is invoked from the CLI command line as:

```
bison mygram.y
```

where 'mygram.y' is the filename of the grammar to be used in creating the parser (the '.y' suffix is traditionally used on UNIX systems to denote a parser grammar).

Bison will create a C file, 'mygram.tab.c', which you can then compile and link with the rest of your program (more details later). Take a peek at the file, but don't expect to make much sense of it.

Bison also accepts two optional arguments which must be given before the grammar name: '-d' will generate a header file 'mygram.tab.h' with token information, and '-v' will generate an output file useful for checking the parse tables. We'll use the first option only; the second requires a greater comprehension of the theory of parsing.

Grammar format

A Bison grammar is divided into three distinct parts: a declarations section, a grammar section and a program section. The two-character sequence '%%' is used to divide the sections. C-style comments delimited by '/*' and '*/' can appear anywhere in all three sections and are ignored by Bison.

The declarations section is used to declare information that both Bison and the generated C file need. This includes any required header files, type definitions, and external function declarations for the actions.

The grammar section lists the rules of the grammar as we described them last issue and the actions to perform for each of those rules.

The program section is optional and can contain any valid C program, which is then included with the parser in the generated C file. Personally I prefer to leave this section blank and code any routines as separate modules.

There are a great many details to cover so let's have a look at our sample grammar.

ECalc: an expression calculator

ECalc is a calculator program with free-form input, floating-point capability, variable assignment, and several built-in functions. Expansion is simple and I encourage you to play around with it.

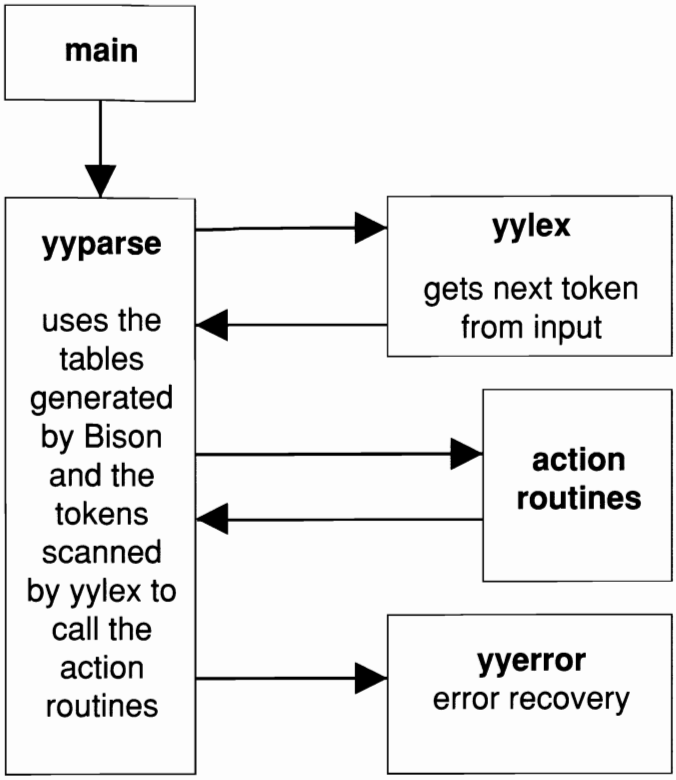


Figure 1

The grammar file for ECalc is shown in Listing 1, the ECalc header file in Listing 2 and the scanner in Listing 3. Two other modules are also needed - these are not listed here due to space limitations, but can be found on the disk for this issue.

The first part of the declarations section for ECalc is delimited by the sequences '%{' and '%}' and will be included unchanged at the start of the C file 'ECalc.tab.c'. The header file 'ECalc.h' includes all the type definitions we (and Bison) need. The functions that are referenced are all used by the grammar actions - Lattice C users might want to expand these to full prototypes.

The '%type' statement is a Bison command used to declare the return type of a nonterminal in the grammar, which we'll discuss later.

The '%token' statements tells Bison which symbols in the grammar are to be interpreted as terminals. This part should never be left out. The statements also declare the return types of the tokens, which we'll also discuss later.

The final statement in the declarations part is the '%start' statement, which tells Bison which nonterminal in the grammar is to be the starting or goal state - parsing ends when this nonterminal is reached. If missing, Bison uses the first nonterminal it encounters in the grammar section.

The grammar section uses the format we discussed last issue with additional support for action routines. Action routines consist of C statements enclosed in braces. The statements are executed when their grammar rule is reduced:

```
statement : expression T_SEMI
                { PrintExpr( $1, TRUE ); }
```

When the parser sees 'expression T_SEMI' it reduces these to 'statement' and then calls the PrintExpr function with two parameters. The '$1' is a special symbol whose use is explained below.

Semantic values

Last issue we differentiated between syntax and semantics. Terminal and nonterminal symbols have both syntactic and semantic values. The token 'T_NUM' in the ECalc grammar stands for an integer or floating-point number, but the semantic value of that token is the actual value of the number. Semantic values for tokens are assigned by the scanner. The semantic value of a nonterminal is handled by the parser and is likely the return value of an action routine. Special symbols are used to access these values.

Consider the rule:

```
primary_expr : T_NUM
                    { $$ = $1; }
```

When the parser sees the 'T_NUM' token it reduces it to 'primary_expr' and invokes the action routine '$$ = $1;'. '$$' is the symbol that represents the semantic value of the nonterminal being defined, in this case 'primary_expr'. '$1' refers to the semantic value of the first symbol on the right-hand side of the rule, in this case 'T_NUM'. If we translate the action into English we get: 'assign the semantic value of T_NUM as the semantic value of primary_expr'. In other words the value of the number we just scanned is accessible to any rule that uses 'primary_expr'.

Say we input the following sequence to ECalc:

```
5 + 3
```

The scanner will code these as:

```
T_NUM T_PLUS T_NUM
```

Eventually the parser will reduce these to the sequence:

```
expression T_PLUS mult_expr
```

This is one of the expansions for 'expression', and the action

```
$$ = Add( $1, $3 );
```

will be invoked. The Add routine will then get the values 5 and 3 passed to it since these are the semantic values associated with 'expression' and 'mult_expr'.

Of course, Bison has no idea as to how these semantic values are encoded. The 'yystype' union defined in 'ECalc.h' declares all the possible types for semantic values:

```
typedef union {
      char   *symbol;
      number  value;
  } yystype;
```

Here a semantic value is either a pointer to a string or a structure ('number' is also defined in the header file). Bison uses this union to reserve enough room in a stack to hold the semantic values. The '%token' and '%type' commands in the declarations section of the grammar file are used to declare which type of semantic value each terminal and nonterminal returns - these ensure that all assignments and function arguments are properly typed.

Semantic values for tokens are passed to the parser from the scanner using the global variable 'yylval' - see Listing 3.

Operator precedence

You might be wondering why all the arithmetic operators in ECalc aren't handled by the same grammar rule. The reason is simple: certain operators have precedence over others, so that 5 + 2 * 3 evaluates as 11 and not as 21. The grammar rules reflect this precedence - lower priority operators are found higher up in the grammar. Try inserting a rule for exponentiation into the grammar, remembering that it has precedence over all operators except the unary minus and expressions in parentheses.

Using the parser

The C file Bison generates consists of a single routine called yyparse(). This routine should be called from main() after any initializations are complete. yyparse() will only return control back to main() when parsing is complete.

The scanner is a routine called yylex(), coded by the user or generated automatically by a lexical analyzer. yyparse() calls the yylex() routine to get the next token. The token values that yylex() returns are defined in the header file generated with the '-d' option. yylex() must also store the token's semantic value in the global variable 'yylval'. yylex() should return a zero when there is no more input to scan. (Note that the parser doesn't care how or where yylex() gets the tokens - from a file, from the keyboard, or even from an Intuition gadget.)

Any action routines used in the grammar must be coded separately by the user. Figure 1 illustrates the organization of a Bison-based compiler.

Error recovery

A syntax error occurs when the parser can't find a rule in the grammar that matches the tokens yylex() has passed it. Bison has two methods of handling syntax errors. The first and simplest is to abort parsing - yyparse() will return the value 1 instead of the value 0 that indicates a successful parse.

The second method involves an attempt at error recovery - Bison will skip some of the tokens in an attempt to find a sequence it can continue to parse. The nonterminal 'error' is reserved for use in the grammar to tell Bison where to try a recovery. In the ECalc grammar the rule

```
statement : error T_SEMI
                { yyerrok; }
```

...tells Bison to skip tokens until it reaches a semicolon (token 'T_SEMI') and to continue as if a 'statement' had been encountered. The action to invoke consists of a special macro 'yyerrok' which tells Bison to ignore the error. The action could conceivably include other error-handling statements as well.

Whether Bison can recover from an error or not, it first passes a string to the routine yyerror() to use in printing an error message. yyerror() is also a user-written function.

Using Bison on the Amiga

Before running Bison execute the following CLI instructions from the Bison directory:

```
copy bison.simple s:bison.simple
makedir ram:t
assign t: ram:t
```

The file 'bison.simple' needs to be in the S: directory and the T: directory is where Bison stores its temporary files.

Run Bison and compile the parser file that it generates. Users of Aztec C 3.6a will not be able to compile what Bison generates without first getting rid of all the references to 'const' in the C file. The easiest way to fix this is to load the C file into your favorite text editor and change all occurrences of 'const' to 'CONST'. (The error occurs because 'const' is defined as a macro - Aztec C doesn't like this.) Aztec users should also use the '+L' (32-bit integers) option when compiling and the 32-bit libraries when linking.

Besides the yylex() and yyerror() routines, the version of Bison I have also requires the use of two UNIX functions, alloca() and bcopy(), neither of which are supplied by the Aztec or Lattice compilers. bcopy() is easily simulated as shown in Listing 3 - it corresponds directly to the Aztec movemem() function. alloca() allocates memory from a function's stack frame - memory which is automatically freed when that function finishes execution. In ECalc I simulate alloca() with a call to malloc(), which isn't as elegant and assumes that the C compiler will free the memory automatically upon program exit. A better fix would be to modify the Bison source code, but this kludge works.

Running ECalc

Instructions for compiling and linking ECalc are included in a README file with the source code and a copy of the Bison executable.

A sample session with ECalc follows:

```
Expression Calculator. Type 'quit' to exit.
  5 + 3;
  ==> 8
  a = 4.0 + 3;
  ==> a = 7.0
  pi;
  ==> 3.14159
  sineofpi = sin( pi );
  ==> sineofpi = 0.0
  intofpi = integer( pi );
  ==> intofpi = 3
  3 * a;
  ==> 21.0
  quit
```

Each command to ECalc must end with a semicolon, except for the 'quit' that ends the session. ECalc will display the result of the expression or assignment. Built-in functions include sin, cos, tan, exp, log, and sqrt (modelled after the C functions of the same name) as well as two conversion functions: 'integer' and 'real'. More detailed instructions are found in the README file. I encourage you to play with ECalc and modify the C files to add more features.

Final Notes

I've tried to cover as much as possible in the limited space available, but as always there are many details that have been left unsaid. A manual for Bison can be obtained from the Free Software Foundation, 675 Mass Avenue, Cambridge, Mass. 02139 U.S.A. FSF distributes Bison as freeware, but please keep in mind that it is not in the public domain. Detailed copyright restrictions are found in the both the source code for Bison (Fish Disk #136) and in the parser file Bison generates.

Information on Bison/Yacc can also be found in many textbooks on compiler construction and in the *UNIX Programmer's Manual*.

Listing 1: 'ecalc.y', the Bison grammar for *ECalc*

```
/*
 * ECALC.Y -- Simple Bison/Yacc grammar for a
 *            calculator program.
 *
 * Grammar and program Copyright (c) 1989 by Eric Giguere
 * Permission granted to distribute free of charge for
 * non-commercial purposes.
 *
 * The Bison parser generator is available on Fish Disk #136.
 * More information on Bison and other GNU (Gnu is Not Unix)
 * freeware products can be obtained from
 *
 *    Free Software Foundation Inc.
 *    675 Mass Avenue
 *    Cambridge, Mass.
 *    U.S.A.  02139
 *    (617) 876-3296
 *
 * This program was orginally published in the
 * Transactor for the Amiga as part of a series
 * on using the Bison/Yacc parser generator.
 *
 */

/*
 * Include type and symbol information for the
 * C file that will be generated.
 */

%{
    #include <stdio.h>
    #include "ecalc.h"

    /* Action routines */

    extern void   PrintExpr(), AssignExpr();
    extern number Add(), Subtract();
    extern number Multiply(), Divide(), Modulo();
    extern number ZeroType(), CallFunction();
    extern number SymbolValue();
%}

/* Define return types for nonterminals */

%type <value> expression mult_expr unary_expr primary_expr

/* Define return types for tokens */

%token          T_QUIT T_SEMI T_ASSIGN T_LPAR T_RPAR
%token <symbol> T_ID
%token <value>  T_NUM  T_PLUS T_MINUS
%token <value>  T_MULT T_DIV  T_MOD

/* Define start symbol */

%start goal

%%

goal            : statement_list T_QUIT

                ;

statement_list  : /* empty */

                | statement_list statement

                ;

statement       : expression T_SEMI
                    { PrintExpr( $1, TRUE ); }
                | T_ID T_ASSIGN expression T_SEMI
                    { AssignExpr( $1, $3 ); }
                | error T_SEMI
                    { yyerrok; } /* Try to recover from error */
                ;

expression      : mult_expr
                    { $$ = $1; }
                | expression T_PLUS mult_expr
                    { $$ = Add( $1, $3 ); }
                | expression T_MINUS mult_expr
                    { $$ = Subtract( $1, $3 ); }
                ;
```

```
mult_expr       : unary_expr
                    { $$ = $1; }
                | mult_expr T_MULT unary_expr
                    { $$ = Multiply( $1, $3 ); }
                | mult_expr T_DIV unary_expr
                    { $$ = Divide( $1, $3 ); }
                | mult_expr T_MOD unary_expr
                    { $$ = Modulo( $1, $3 ); }
                ;

unary_expr      : primary_expr
                    { $$ = $1; }
                | T_MINUS unary_expr
                    { $$ = Subtract( ZeroType( $2 ), $2 ); }
                ;

primary_expr    : T_ID
                    { $$ = SymbolValue( $1 ); }
                | T_NUM
                    { $$ = $1; }
                | T_ID T_LPAR expression T_RPAR
                    { $$ = CallFunction( $1, $3 ); }
                | T_LPAR expression T_RPAR
                    { $$ = $2; }
                ;
%%
```

Listing 2: 'ecalc.h', the header file for *ECalc*

```
/*
 * ECalc -- Copyright (c) 1989 by Eric Giguere
 *          Published in Transactor for the Amiga
 *          as part of the Bison/Yacc tutorial.
 *
 *  ECALC header:  Common definitions
 */

#define BUFFER_SIZE 256

#define INTEGER_TYPE 1
#define REAL_TYPE    2
#define ERROR_TYPE   3

#define NULLCHAR '\0'

#ifndef FALSE
    #define FALSE 0
    #define TRUE  1
#endif

/* These three types are used by the parser as well */

typedef union int_or_real
  {
    long   integer;
    double real;
  }
    int_or_real;

typedef struct number
  {
    unsigned char class;
    int_or_real   value;
  }
    number;

typedef union       /* Semantic value */
  {
    char   *symbol;
    number  value;
  }
    yystype;

#define YYSTYPE yystype

#define T_NULL  0
#include "ecalc.tab.h" /* Get token definitions */
```

Listing 3: 'yylex.c' mainline and scanner for *ECalc*

```
/*
 * ECalc -- Copyright (c) 1989 by Eric Giguere
 *          Published in Transactor for the Amiga
 *          as part of the Bison/Yacc tutorial.
 *
 *  YYLEX module:  Mainline & scanner.
 */

#include <stdio.h>
#include <ctype.h>
#include "ecalc.h"

YYSTYPE yylval;

extern long   atol();                  /* Convert string to long */
extern double atof();                  /* Convert string to double */
extern void   *malloc(), free();
extern void   InitTable(), FreeTable(); /* From SYM module */

static int  EndofScan;            /* TRUE when 'quit' is found */
static int  CurrChar;             /* Current input character */
static char Buffer[ BUFFER_SIZE ]; /* Input buffer */

/* Forward references */

static int ScanNumber(), ScanSymbol(), ScanOperator();


/* Program mainline */

int main()
  {
    CurrChar  = ' ';
    EndofScan = FALSE;
    InitTable();

    puts( "Expression Calculator.  Type 'quit' to exit." );
    yyparse(); /* Invoke parser */

    FreeTable();

    return( 0 );
  }

/* Scanner -- invoked from yyparse() */

int yylex()
  {
    int token;

    while( EndofScan == FALSE && CurrChar != EOF )
      {
        if( isspace( CurrChar ) ) /* Ignore white space */
          {
            CurrChar = getchar();
            continue;
          }
        else if( isdigit( CurrChar ) ) /* Number */
            return( ScanNumber() );
        else if( isalpha( CurrChar ) ) /* Symbol */
            return( ScanSymbol() );

        /* Must be a operator */

        token = ScanOperator();
```

```
        if( token != T_NULL )
            return( token );
      }

    return( T_NULL ); /* Tell parser to quit */
  }


/* Collect an integer or floating-point value */

static int ScanNumber()
  {
    char *ptr;

    /* Collect integer part of number */

    for( ptr = Buffer; isdigit( CurrChar ); CurrChar = getchar() )
        *ptr++ = CurrChar;

    *ptr = NULLCHAR;

    if( CurrChar != '.' ) /* Return an integer */
      {
        yylval.value.class         = INTEGER_TYPE;
        yylval.value.value.integer = atol( Buffer );

        return( T_NUM );
      }

    /* Collect fractional part of number */

    *ptr++ = CurrChar;

    for( CurrChar = getchar(); isdigit( CurrChar ); CurrChar = getchar() )
        *ptr++ = CurrChar;

    *ptr = NULLCHAR;

    yylval.value.class      = REAL_TYPE;
    yylval.value.value.real = atof( Buffer );

    return( T_NUM );
  }

/* Collect a symbol name */

static int ScanSymbol()
  {
    char *ptr;

    for( ptr = Buffer; isalnum( CurrChar ); CurrChar = getchar() )
        *ptr++ = CurrChar;

    *ptr = NULLCHAR;

    /* QUIT is a special case */

    if( strcmp( Buffer, "quit" ) == 0 )
      {
        EndofScan = TRUE;
        return( T_QUIT );
      }

    /* Allocate space -- don't forget to free it later on! */

    yylval.symbol = malloc( strlen( Buffer ) + 1 );
    strcpy( yylval.symbol, Buffer );

    return( T_ID );
  }

/* Collect operator */

static int ScanOperator()
  {
    int ch;

    ch       = CurrChar;
    CurrChar = getchar(); /* read in next character */

    /* This switch could also be implemented quite easily
       as a table */

    switch( ch )
      {
        case ';':
            return( T_SEMI );
        case '(':
            return( T_LPAR );
        case ')':
            return( T_RPAR );
        case '+':
            return( T_PLUS );
        case '-':
            return( T_MINUS );
        case '*':
            return( T_MULT );
        case '/':
            return( T_DIV );
        case '%':
            return( T_MOD );
        case '=':
            return( T_ASSIGN );
        default:
            printf( "*** Unexpected character '%c' in input ignored\n", ch );
            return( T_NULL );
      }
  }

/* Error routine called by yyparse() -- we purposely
   ignore the error string and print our own */

yyerror( str )
  char *str;
  {
    puts( "*** Please re-enter your input" );
  }

/* Simulate a UNIX memory allocation function
   that Bison uses */

void *alloca( size )
  long size;
  {
    return( malloc( size ) );
  }

/* Simulate a UNIX memory copying function */

void bcopy( from, to, size )
  char *from, *to;
  long size;
  {
    #ifdef AZTEC_C
        movmem( from, to, size ); /* only Aztec */
    #else
        while( size-- > 0 )
            *to++ = *from++;
    #endif
  }
```

A Matter Of Style

How to write readable C, and why it matters

by Ben Blish

Ben Blish is president and founder of Black Belt Systems, a Montana-based Amiga software development company. Among the Amiga products he has developed are the electronic-CAD programs PCLO and PCLOplus. He is currently involved in a number of hardware-oriented projects for the Amiga.

This article is an attempt to address the problem of the highly disparate styles in use in Amiga C language programs.

I hope to present a series of concepts that will serve as a basis to structure your C source language formatting in such a way as to accomplish the following:

1) Enhance maintainability for the author of the source code
2) Enhance portability among Amiga C language types
3) Enhance clarity for readers who are *not* the author of the source

The first section of this article deals with issues that do not affect the actual code generated by your source - only the readability of the source, both while generating it and while rereading it. The second part of the article deals with an issue that does affect the executable code your source generates. If you elect to utilize those suggestions, you will need to change the strategy used in your software.

Over quite a few years of C coding, Timothy Keating and I have slowly created a series of formatting conventions that, when taken together, result in C source code that we believe goes a long way towards meeting the three goals stated above. These conventions are quite general, and most often apply to one or both of two main components of source language formatting: indentation, and usage of the curly brackets, the { and } symbols.

IF and WHILE clauses

Here is how we think these should be constructed:

```
if (condition)
   {
      single_statement();
   }
```

```
if (condition)
   {
      statement_1();
      statement_2();
   }

while(condition)
   {
      single_statement();
   }

while(condition)
   {
      statement_1();
      statement_2();
   }
```

These four examples conform to the following rules:

1) Any single or multiple statement clause dependent from a conditional is enclosed in brackets.
2) When bracketing, a constant (to your taste) amount of indentation is used for the brackets themselves.
3) Any clause within brackets is indented the *same* set amount as the brackets are, beyond the brackets themselves.
4) Brackets are never used in the same line with a statement.
5) Opening and closing brackets are always at the same level of indentation.

These rules, when used consistently, yield a number of impressive benefits without sacrificing code structure, efficiency, or clarity.

First, if code is to be modified - for instance, if you have to add a statement to a clause - use of the recommended structure tells you exactly where to put the new statement; the construction below is misleading and can cause problems, and requires modification of the conditional structure itself.

The fragment shown next is not extensible - if you add a statement to it at any point, the grouping of the other statements will be modified. Nor would the mere addition of indentation solve the underlying problem.

```
if (condition)
while(condition)
if(condition)
statement();
```

In contrast, the following construction (a) clearly, visually, helps you determine where the statement should be placed; (b) requires no modification; and, (c) is extensible - no part of the conditional structure need be modified to add any number of statements to any part of this complex conditional.

```
if (condition)
   {
      while(condition)
         {
            if (condition)
               {
                  statement();
               }
         }
   }
```

Note the clear alignment of both bracket levels and operative statements. You can visually, almost instantly, determine if you have a problem with the logical grouping of your construct(s), because if you do, the brackets and the text will not line up vertically. Adding a statement to any section of this example is simple, and highly intuitive - it's quite clear which level of the *if-while-if* construct it will belong to.

An important component of the strategy used here is the completely separate, but dependent, indentation levels occupied by program statements and the curly brackets. When using this technique, you can always determine where a logical group ends - even if it's on another page - without reading the program code. You can always find the brackets easily, because they are never embedded in text. They stand out, actually giving program clauses a pronounced visual identity. Another important advantage only becomes apparent when you make a mistake in opening/closing bracket pairs. If you fail to close a set of brackets, your indent level will be misaligned from its proper position by the indent used for the brackets and the indent used by the program statements - it stands out like the proverbial sore thumb.

Look at the following example; there is a logical problem in it, which you can *see* without reading the program source at all, due to the bracketing conventions used. Try to bear in mind that you are not supposed to be reading the program source - you are looking at the listing as a *graphic* object, which should conform to a specific set of expectations.

```
function()
   {
      statement();
      while(condition)
         {
            if (condition)
               {
                  statement();
                  switch(index)
                     {
                        case ONE:
                           {
                              break;
                           }
                        default:
                           {
                              break;
                           }
               }
         }
   }
```

Here's how we think *switch/case* constructs should be built:

```
switch(index)
   {
      case ONE:
         {
            break;
         }
      case TWO:
      case THREE:
         {
            break;
         }
      default:
         {
            break;
         }
   }
```

Exactly the same bracketing rules apply here, and for very similar reasons. However, if you wish a case to fall through, you may omit the *break;* for that particular case. Otherwise, the format is exactly identical to that shown. *Case* statements that are not bracketed perform a function similar to a label/*goto* pair, with the added convenience that the compiler looks up which *goto* to use. When you add the bracketing convention shown here, the individual cases serve, in concert with the brackets, to identify and group the clause you want executed when that case occurs - again, it becomes a visual part of the source code, not something you have to read every line of to understand. The following construct suffers from the same problems as the closed-up *if-while-if* example shown earlier:

```
switch(index) {
case ONE:
break;
case TWO:
case THREE:
break;
default:
break;
}
```

It offers no *visual* clue as to the logical intent of the programmer, and there is no 'help' available as to whether a statement belongs where it is - the logical clauses do not stand out; the code must be read to determine exactly where they are. Problems like these, which are obvious when shown with just a few statements, are much more severe in complex sections of code. Suppose you need to add a statement that is dependent on the *if* leading this block, but that occurs after the *switch*:

```
if (condition)
switch(index)
case ONE:
case TWO:
break;
case THREE:
break;
```

You'll need to restructure the entire block, because the *if* loses scope at the end of the *switch*. With a long *switch*, you can be easily misled because the *if* may be on another page - there is no visual clue as to where its scope ends. If you use the bracketing conventions presented here, not only will you know exactly where the scope passes from the *switch* to the *if*, but you will also never be bitten by a statement that is in the wrong (or completely outside of any) dependent clause. Using the proposed method, you build the fragment in an ordered fashion, as follows:

Step 1
```
if (condition)
  {
  }
```

Step 2
```
if (condition)
  {
    switch(index)
      {
      }
  }
```

Step 3
```
if (condition)
  {
    switch(index)
      {
        case ONE:
          {
            break;
          }
        case TWO:
        case THREE:
          {
            break;
          }
        default:
          {
            break;
          }
      }
  }
```

Notice that each section of the source had its opening and closing brackets completed before the dependent clause was written. That simple procedure provides a high degree of assurance that dependent clauses are logically complete, subordinate to the correct enclosing structure, and indented properly. The second of these three points deserves some additional discussion before we continue.

If you write a program fragment such that you later need to go back and make it dependent upon a conditional, it may be difficult to clearly identify the beginning and end of the fragment. Because it wasn't written as a subordinate member, it isn't bracketed or indented. On the other hand, if you write the complete conditional first - even if you don't fill in the condition - you have clearly defined the structure such that it is highly unlikely you can make a mistake if you simply watch your indent levels. You can write it like this:

Step 1
```
if ()
  {
  }
```

Step 2
```
if ()
  {
    statement_1();
    statement_2();
  }
```

Then go back and fill in the conditional later. This completely avoids paging through the source code, reading it line by line to find out just where you should delineate the dependent clause. It's worth pointing out that if you elect to use the indentation methods outlined here, there would be an additional job involved as well - you would need to go through each line of the newly dependent clause and indent it to match its level of dependency. This isn't too onerous a task, especially if you have a sophisticated editor like Uedit (yes, that's a blatant plug - best editor I've ever seen, and I think I've seen all of the better Amiga editors); however, it is an extra job, which to our way of thinking makes it something to avoid.

Now, if you need to add the statement mentioned in the initial example, it goes before the last bracket, indented exactly between the two closing brackets - and you don't need to examine the intervening code to make certain, either. You don't have to modify the structure at all - again, we have an extensible construct. This is one of the keys to efficient modification and maintenance of existing source code. Using this technique allows you to be able to utilize pre-existing control structures to the fullest, without changing the original scope of those structures.

```
if (condition)
  {
    switch(index)
      {
        case ONE:
          {
```

```
            break;
         }
      case TWO:
      case THREE:
         {
            break;
         }
      default:
         {
            break;
         }
   }
   New_Statement();
}
```

Perhaps by now you've detected an underlying theme: we feel very strongly that a large number of visual cues are the basis for the best coding style. The less cues, the poorer the style.

In simple terms, the time you spend trying to figure out what a previously written clause does is time you could have used to make the program more powerful, or complete; instead, you're essentially reinventing your own code.

When you consider maintenance of your code by someone else, this becomes even more important. *You* may be able to remember every nuance of indentation (or lack of it) of each code fragment, but others don't share your past experience (writing it) and so are left out in the cold. This method provides them with a good start on the program structure - they can *see* it!

Return value casting

Many times, you'll see code in which casts are scattered hither and yon, peppering the source as if they did something meaningful. Well, sometimes they do, but usually they just serve to repeatedly fix something that is broken in the beginning of the module.

The reason we focus on the use of casts is because the situation we're going to outline is very common. It unnecessarily complicates individual lines of source code, something that should always be avoided.

You may have run across the common Intuition call *OpenWindow( )*. It's a very straightforward call requiring as a parameter a pointer to a *NewWindow* structure, and returning a pointer to a *Window* structure if it succeeds in opening the window, or a zero if it fails.

Most often, you'll see it used like this:

```
if (!(MyWindow =
      (struct Window*)OpenWindow(&NewWindowStructure)) )
   {
      /* Window didn't open! Do something! */
   }
```

If you place this line at the beginning of your module:

```
struct Window *OpenWindow();
```

then you can write the *OpenWindow( )* call like this:

```
if (!(MyWindow = OpenWindow(&NewWindowStructure)) )
   {
      /* Window didn't open! Do something! */
   }
```

It's worth doing, because in a particular source file you may have a number of calls to *OpenWindow( )*, all of which are much more readable, easier to enter in the first place, and less subject to typos and so on because there was less to be typed.

Take the time to find the definitions of each external call you're going to use in a module, and make sure that they are in place at the beginning of the source code. It'll take you less time to write the modules, and they'll be easier to create, read, and maintain.

Some compilers may come with header files that define some subset(s) of the available external routines. If your compiler has these, just include them at the front of the module.

Pulling it all together

In order to provide a better feel for the rules presented here, you may wish to examine the following module. It is 'real-world' Amiga code, written to conform to the concepts presented here while doing a useful Amiga task (handle Intuition messages to any window) in an efficient, concise manner. Notice the way that the incoming variable and the local variables co-exist 'around' the function's opening curly... it's done that way so it looks different, because it *is* different. Variable declarations are neither conditional nor executable statements.

```
/* so we don't have to cast in the source - keeps source clean */

struct IntuiMessage *GetMsg();

/* handles intuition messages for a window;          */
/* entered with pointer to window in question, and   */
/* numeric window ID for subsequently called modules */
/* that may need to know which window this is.       */

extern char have_console; /* indication of program's I/O resources */

void Window_Handler(win_ptr,win_id)
  struct Window *win_ptr;
  int win_id;
  {
  char                buffer[50];
  struct IntuiMessage *msg;
  struct MsgPort      *port;
  struct Gadget       *gad;
  char                truth;
  unsigned long       mask;
  int                 code,class,qualifier;
    truth=1;
    port = win_ptr->UserPort;
    mask = 1 << port->mp_SigBit;
```

```
  while(truth)
    {
      Wait(mask);
      while( msg=GetMsg(port) )
        {
          code      = msg->Code; /* capture useful message data */
          class     = msg->Class;
          qualifier = msg->Qualifier;
          gad       = msg->IAddress;
          ReplyMsg(msg); /* and return message immediately */
          switch(class)
            {
              case MENUPICK: /* User selected a menu item */
                {
                  menu_proc(code,win_ptr,win_id);
                  break;
                }
              case GADGETUP: /* User hit a gadget */
                {
                  gadget_proc(gad,win_ptr,win_id);
                  break;
                }
              case RAWKEYS: /* User pressed a key */
                {
                  key_proc(code,qualifier,win_ptr,win_id);
                  break;
                }
              case CLOSEWINDOW: /* terminate window handler */
                {
                  truth=0;
                  break;
                }
              default: /* process surprises here */
                {
                  sprintf(buffer,"Unknown IntuiMessage in Window %d",win_id);
                  if (have_console) /* inform user via CLI */
                    {
                      printf("%s\n",buffer);
                    }
                  else /* inform user via WB request window*/
                    {
                      do_requester(win_ptr,buffer,"","-OK-","-OK-");
                    }
                  break;
                }
            }
        }
    }
}
```

Now, that module could have been written without indentation, and even more simply, in the following way:

```
extern char have_console; /* indication of program's I/O resources */

  void Window_Handler(win_ptr,win_id)
  struct Window *win_ptr;
  int win_id;
  {
  char buffer[50];
  struct IntuiMessage *msg;
  struct Gadget *gad;
  char truth;
  int code,class,qualifier;
  truth=1;
  while(truth) {
  Wait(1 << win_ptr->UserPort->mp_SigBit);
  while(msg=(struct IntuiMessage)GetMsg(win_ptr->UserPort)) {
  code=msg->Code; /* capture useful message data */
  class=msg->Class;
  qualifier=msg->Qualifier;
  gad = msg->IAddress;
  ReplyMsg(msg); /* and return message immediately */
  switch(class) {
  case MENUPICK: /* User selected a menu item */
  menu_proc(code,win_ptr,win_id);
  break;
  case GADGETUP: /* User hit a gadget */
  gadget_proc(gad,win_ptr,win_id);
  break;
  case RAWKEYS: /* User pressed a key */
  key_proc(code,qualifier,win_ptr,win_id);
  break;
  case CLOSEWINDOW: /* terminate window handler */
  truth=0;
  break;
  default: /* process surprises here */
  sprintf(buffer,"Unknown IntuiMessage in Window %d",win_id);
  if (have_console) { /* inform user via CLI */
  printf("%s\n",buffer); }
  else {/* inform user via WB request window*/
  do_requester(win_ptr,buffer,"","-OK-","-OK-"); }
  break;} } }
  }
```

Can you read it? More importantly, could you *change* it quickly? If you can read it, then is the program flow clear?

I have to mention here that I have seen code considerably worse than this in example source downloaded from CompuServe's data libraries, and on various PD disks; worse yet, some of that source was from some of the Amiga community's most respected turbo-techies.

Of course, many programming styles fall between the two examples shown here. Some fall outside them - both ways - better and worse, but these are few and far between. Some programmers indent some things, sometimes; others indent for no particular reason. Some cast everything in sight and others never cast, letting the warnings flow down the screen like sewage into some forgotten swamp (how colourful...).

Even if you elect to use some other methodology (which is likely, since C coding style is a religious issue), please, please, maintain some consistency, and don't allow source that generates warnings to be released as example code!

Think of the new programmer. Warnings scare them, because they don't really understand the mechanisms that cause them. Make certain your code compiles without any threatening messages before you release it on some starry-eyed Modula devotee looking into C for the first time.

ILAC - A terminal disease

There is one particular coding practice that serves as the main barrier to both inter-machine and intra-machine portability, readability, and maintainability. We know of several such practices, but this one stands out as the worst offender by a huge margin.

This is the use of in-line assembly code. Some compilers, notably the Manx Aztec C Compiler for the Amiga, allow the use of 68000 assembly language constructs within C source code: you can switch to assembly language at virtually any point in

your C source and begin a hand-coded sequence of any length. This is done (presumably) to attain code efficiency, accuracy, or execution speed that the compiler itself is incapable of.

There are myriad reasons to avoid this practice; so many, in fact, that an entire article could (and probably should) be written to address them. We'll just point out some of the more obvious problems here.

First and foremost, in line assembly code (we'll call it ILAC for rest of this discussion) is pure death for portability across machine types. Got a terrific application? Perhaps you'll be approached to put it on a great new RISC machine - 256,000 colours, 32-bit blitter, 16-bit 16-channel sound, etc, etc, ad nauseam. ILAC? Uh-oh. You can forget that code working, you're going to have to completely rewrite the module(s) that use it.

Perhaps you think you'll never leave the Amiga's warm and safe environment. All right, then consider this: suppose someone - Lattice, for example - comes out with a truly superb new C compiler. This thing has the built-in source level debugger you've come to depend on, it's conformant with the new ANSI standard, and makes better code than your current compiler. ILAC? Same problem, different day. Your code will *not* work. The low-level register assignments and dependencies are different, and in any case ("At least, not yet", as Obi-Wan Kenobi once muttered) the Lattice compilers don't support ILAC anyway.

Too bad for you - while the competition, who were wise enough to write their assembly source in modules for an assembler, linking it in separately, already have their competing application up and running on the new compiler. Your application is better, as you know... but since you can't port it, no one else is going to find out.

Next we look at someone else's ability (or lack of it) to understand what you have written. You show them your module, containing ILAC. They say, what's that? You then proceed to explain to them which registers are set up with what variable(s) by the compiler, which you assume have been set by the system, which you are passing back into your C code, and how they get there.

By the time you finish explaining the underpinnings of the ILAC section, the listener may well no longer remember what the rest of the module did.

There is a valid issue to be addressed here, that of performance. Of course, you want performance - users want performance, programmers want performance, salesmen want performance - but ILAC is a lousy way to get it. You're hurting yourself and the end user if you use ILAC.

What can you do instead? If you must write a section in assembly, then do so - but write it as a module - make sure it is callable by using the standard Amiga stacking conventions, and design the module such that all the high performance code is within it. This especially includes loop control and so on - the 68000, which is probably what you are writing for, has some very powerful loop management constructs, notably the DBcc instruction group and the bit set/clear/test instruction groups.

Rule:

Don't use your C compiler as a 'loop control generator'. It's not worth the work you save writing the loop control in assembler, and can cost you many hours of work later, should the requirements of the application change.

If code of this type ever needs to be ported to another machine, like our mythical RISC machine (you wouldn't port down to a clone machine, would you?) There is no problem - because the assembly module is a defined quantity, with a specific set of inputs and outputs. You may be able to regenerate the module in C, if the machine you are porting to has enough of an edge in power. If not, you have still reduced the problem from one of recoding an application where ILAC is scattered thoughout the source, requiring recoding of much of the application, to one of recoding a module.

When you talk about portability within the Amiga environment, you gain much more - the assembly language modules don't have to be rewritten at all - and the C modules may not have to be either, since they aren't chock full of references to what after all is another langauge.

Consider again the case of source code released for instructional purposes. The Amiga boasts a huge public domain library of C source code. Unfortunately, much of it is unusable because of the inclusion of ILAC sections. Here is a terrific resource that goes to waste in many cases, because it won't compile, it won't link, and it just plain won't *work* unless you have exactly the same compiler environment as the author did.

Remember, too, that users of another compiler do not have the documents that explain the assumptions used by your compiler relative to ILAC. They have no way to find out which registers are loaded with what. And as a semi-serious aside, much of this type of hacking occurs in the wee hours - there isn't even anyone you can call under those circumstances!

On the other hand, source code written without ILAC sections works for everyone - the difference is quite clear.

The bottom line is: when writing C - *only* write C. When writing assembler, *only* write assembler. Let the two meet at the call/return module level, never mix them together in the same source module.

In conclusion, I would like to say that both Tim and I would be very interested to hear about new ideas and concepts for extension and modification of the ideas presented here. They are by no means 'written in stone', and we're certain that some of you could contribute considerably to them. ◘

Three Hard Disk Back-up Utilities

A tight race with no clear winner

Review by Steve Ahlstrom
Copyright © 1989 Steven Ahlstrom

One of the fastest growing segments of the Amiga peripheral market is the hard disk. Until recently there were only a couple of manufacturers of Amiga hard disk controller cards. Now there are so many one can lose count!

Adding a hard drive to an Amiga transforms the machine. If you do not own one you may be under the impression that you do not need one. If you only knew. When you add a hard drive, it is like getting a brand new computer.

Buying a hard disk solves problems: no more looking through stacks of disks for the program you want to run. (If you are like me, half or more of those floppies are unlabelled.) And the increase of speed in reading and writing is an appreciated pleasure in itself.

However, owning a hard disk is not all roses - there are a few thorns thrown in. The biggest thorn of all is the problem of making sure all that software you've put on your hard disk will survive if you should have a sector go bad. Or maybe turn your computer off during a write operation. Or a half dozen other things that might go wrong.

Lately I've been looking at three different hard disk back-up programs. The first program is the best known. The second is one of the earliest for the Amiga. The third is a brand new entry into the marketplace. They all have strengths and weaknesses. None of the programs is copy protected, none will interfere with multitasking, and all will run on an unexpanded (512K) Amiga. I am going to be up front and tell you that, in my opinion, none is better than the others. Read on and find the one that meets your needs.

Quarterback

Central Coast Software, 424 Vista Avenue, Golden, CO 80401, (303) 526-1030. Suggested retail price: $69.95 US.

Quarterback's biggest advantage is that it will back up any AmigaDOS device, which includes Bernoulli boxes, streaming tape devices, high density floppy drives, jh0:... essentially all of those non-trackdisk.device peripherals.

Using Quarterback is simple. It boots up into a full sized Workbench window with two gadgets showing: *Backup Files* and *Restore Files*. When you click on *Backup Files* you are prompted for the hard disk device name and optional subdirectory path. You are also prompted for the primary and alternate destination devices. Quarterback will back up on two floppies, alternately, which cuts down on overall back-up time. When the back-up to your primary device is completed, Quarterback will automatically start writing to the secondary device. In the meantime, you put a new disk into the primary device. This way Quarterback never skips a beat.

Quarterback will build a list of file names in your back-up path. The building of the list up front is what gives Quarterback some rather impressive timings later when you actually do the back-up. ExpressCopy and Saf-T-Net build the list 'on the fly'. This really needs to be taken into consideration if your primary concern is speed of back-up. It's all relative - time is time, whether that time is taken before or during the back-up phase. Be leery of benchmarks based only upon the time needed to write data to a disk.

After building the list, Quarterback shows you the number of files it will back up, the total byte count, and how many floppies will be needed. You are now given options that let you include or exclude files by various criteria: name, date, whether the archive bit is set, or 'all'. You can select a subdirectory in your back-up path and apply the inclusion/exclusion criteria to the subdirectory rather than the entire back-up path. This allows for flexible control over the data you want to archive.

Next, you are presented with various options to set: write with verify, overwrite AmigaDOS disks, beep between disks, set archive bits, and others. The most important of these options are the AmigaDOS disk warning and the write with verify features.

The overwrite feature, in its default mode, will check if the inserted floppy is a valid AmigaDOS disk. Should a disk be inserted that you do not want to overwrite, a warning will be provided. This is a nice safety feature. You are given the option of continuing or inserting a new disk. When this option is turned off, Quarterback does not check.

The other important option is the creation of your back-up with, or without, write verify. With verify, Quarterback reads what it wrote to ensure there are no errors. A reason for backing up your hard disk may be insurance against future hard disk failure. With verify, you can trust your back-up set to be valid. Backing up your data with verify does take significantly more time, but is worth it for the security of knowing that your data is intact. It's a matter of time. Do you take the time now to back up with verify, or find out later you must spend a lot *more* time recreating your lost data?

Once the initial back-up set is made, you can create incremental back-up sets in the future. This means you will only back up the files that were added since your initial hard disk back-up. You can base the selection of files for the back-up set either upon the datestamp or upon the state of the archive bit.

The restoration process is very similar to the back-up process - but backwards. You have all the same options for inclusion and exclusion of directories and files. During the restoration phase you also have the option of not restoring files that have been deleted from your hard disk since the back-up was made.

Quarterback keeps a list of the directories and files it has backed up on the first and last disks of your back-up set. Having copies of this list on two disks is a safeguard against one of them going bad. Quarterback does not write in an AmigaDOS format. Without its own 'table of contents' intact it would be difficult to restore your data.

Quarterback allows you to create an 'Archive Report'. This is a list of the name, date/time stamp, and size of each of the files it has backed up. You may direct this list to be output to your printer or to a disk file. The Archive Report cannot be used in lieu of the catalogue of files written to the first and last back-up disks should these go bad, however.

While the disks that Quarterback creates are not AmigaDOS compatible, you can copy these disks with the AmigaDOS Diskcopy program. You cannot restore using AmigaDOS. Quarterback writes 901k of data to a 3-1/2" floppy and does no file compression.

saf-T-net

Metadigm, Inc., 19762 MacArthur Blvd., Suite 300, Irvine, CA 92715. Suggested retail price: $49.95 US.

saf-T-net was one of the earliest entries into the Amiga hard disk back-up market. It has been plagued by poor marketing and has not received the attention it should have. The program was originally marketed by RSN Software, a bootstrap operation that quickly ran up against a Catch-22 situation: they could not afford to advertise unless sales were strong, yet sales had no chance of being strong if no one knew about the program. Eventually, RSN decided its experiment in the Amiga marketplace was a failure, and the marketing rights to saf-T-net reverted to its author, Bob Rakosky.

The future of saf-T-net looks positive. It has recently been given a facelift with the addition of many new features. Metadigm (of MetaScope fame) has licensed the product for distribution. The Metadigm version should be on your dealer's shelves about the time you read this.

When you bring up saf-T-net you are presented with a custom screen full of gadgets. From here you choose all of your options - and there are a lot of them: back-up or restore, the full name of the device/directory to back up, whether you want a full back-up, incremental back-up, restore, etc. All of the configuration options are available on this screen.

After you have set the various options you want, click on 'OK'. You are shown a new screen - this is where all the action takes place. You start your back-up/restore from here. You are shown the full path name of each file as it is processed. Should you decide the options you have set are not the ones you want, you can back out and return to the configuration screen.

Unlike the other programs, when you want to do an incremental back-up with saf-T-net, you simply add to your existing back-up set instead of creating a new one. This may be a minor point but it *is* convenient. Your back-up disks are numbered by the program. When you want to create an incremental back-up, saf-T-net asks you for the last disk in your back-up set. It then reads in the catalogue information, and starts writing to that disk (if there is space). It is suggested a full back-up be done once a month with an incremental back-up at least weekly. With this approach you only have one set of backup disks to maintain. With other programs you have at least two, and sometimes many, separate sets of back-up disks.

Backing up with saf-T-Net is not as speedy as with QuarterBack. saf-T-Net does not offer a choice of verify and no-verify modes - it always verifies its writes. Should a disk error occur while writing to the back-up disk, saf-T-net will attempt to format the track, verify it, and continue its operation.

saf-T-net does not keep its catalog of files on your back-up disks; it can be saved anywhere you like. This gives you an extra bit of insurance should one or more back-up disks go bad. As mentioned earlier, Quarterback keeps its catalogue on the first and last disk of the back-up set. While it is unlikely that both the first and last disk would go bad, if they do, your back-up set is worthless - you can't restore without the catalogue. With saf-T-Net, you can write the catalogue out to a floppy or to hard disk. Writing it out to hard disk is not recommended - if the hard disk develops problems you could lose the file. However, using this approach allows you to keep more than one copy of the catalogue, depending on your state of preparedness (or paranoia). Even without the catalogue, you can restore from your back-up disks. Also, a new catalogue can be created from your back-up set.

saf-t-net does not write in AmigaDOS format. It packs 907.5 kilobytes of data onto a diskette. It does use compression on

some files, but the type used really is not very significant, giving five to ten per cent compression over the entire back-up set. Compression is time intensive, so the author only uses it when there would be little or no time penalty. Unlike Quarterback, saf-T-net back-up disks cannot be copied with the AmigaDOS Diskcopy program. A program called saf-T-copy is included that will allow you to copy your back-up set.

saf-T-net is a very solid program. It is not the fastest nor the most feature-laden, but it gets the job done. Some of the unique features it does have are there for user safety. Its main goal is to help protect your data.

ExpressCopy

ExpressWay Software, P.O. Box 10290, Columbia, MO, 66205-4405, (314) 474-2984. Suggested retail price: $44.95 US.

ExpressCopy (XCO) is the new kid on the block. It is very feature-packed. Written by Glenn Nielsen of ExpressWay Software, it is sure to become a strong contender.

The most obvious difference with XCO is that it writes AmigaDOS format files. Should you desire to use the new 1.3 Fast File System (FFS) on your floppies, XCO will accommodate you.

Using FFS on floppies is not 'approved' by CBM, but it will give you a nominal increase in both access speed and storage capacity. Normally, to use FFS on floppies you must create a mountlist entry and mount your floppy (an example mountlist is provided in the XCO manual). An AmigaDOS Diskchange command must be issued after inserting a new FFS-formatted floppy - AmigaDOS does not automatically recognize it. XCO does. If you wish, XCO goes so far as to format the disks in FFS format for you while you are creating your back-up - even if you do not have the mountlist entry or have not mounted the FFS drive! To restore from these FFS disks, however, you must set up the mountlist and mount the drive.

By creating AmigaDOS files, XCO has a lot of advantages over Quarterback and saf-T-net. Some of these advantages are double-edged, though, and it isn't easy to decide which method has the most going for it.

On the plus side, the back-up disks created by XCO have the same directory paths and files as the source directory. Should the hard disk fail for some reason, the files backed up using XCO could be used directly from floppy. Imagine this panic situation: Your hard disk becomes invalidated and nothing you do will revive it short of re-formatting. So, you format it with the idea of restoring from your backup set. Half-way through the restore you come up with a bad floppy. What do you do? With either Quarterback or saf-T-net you are pretty much out of luck: you will lose the files (unless you have a back-up of your back-up set!). With XCO, you stand an excellent chance of restoring the disk with either DiskSalv or DiskDoctor. Using AmigaDOS format increases the security and integrity of your back-up.

Another advantage of this approach is that you can use normal AmigaDOS commands or your favourite directory utility to restore. XCO comes with a separate restore program, but the only time you really need to use it is when you have backed up files that are larger than the capacity of the floppy. Restoring with AmigaDOS will not help you there.

Using AmigaDOS format also has its downside, however. The main disadvantage is that you store less data per disk. Quarterback writes 901K to each disk. saf-T-net writes 907.5K. With XCO, it's variable. At best, it can write a bit over 800K per disk, sometimes much less. If you had filled a disk to 500K, there would still be a little over 300K of data space free. If the next file to back up is 400K in size, XCO will ask for another disk in the set and effectively leave the previous one only partially filled. Depending on the luck of the draw - that is, the distribution of file sizes on your hard disk - XCO will require more disks for the back-up than the others. Even if every disk is filled to capacity, XCO will still use more disks because each disk, filled to capacity, holds less data. ExpressWay has said that in a future update they will attempt to utilize the 'wasted' space by holding back the write of the large file, writing other smaller files in the directory until either the target disk fills or it is time for the next disk to be inserted.

You might think that XCO would also be at a speed disadvantage - that it would take longer to create the AmigaDOS files. That's not the case, however. In fact, XCO is very competitive timewise (refer to the chart at the end of the article).

Like the others, XCO has a full complement of options for backing up and restoring. You can include/exclude files by the archive bit, date stamp, and filename pattern matching.

XCO will automatically detect if the disk inserted is a valid AmigaDOS disk. If so, a warning will be given, and you will determine whether to write over it. This helps protect from inserting a disk you do not want to lose, such as one of the disks you just completed in your back-up.

XCO gives you a lot of control over its use. Not only is it fully multitasking but, should other programs be active while running XCO, you can easily change the priority level. Should you need temporarily to use the same devices XCO is using, you can click on the 'Halt' button. XCO will stop after it has completed writing the current file and let you use the drive for another program. When done, click on 'Resume'. XCO will check that you have re-inserted the correct disk(s), then resume the back-up. Should XCO come across a file or directory that it cannot access because another program has a lock on it, you will have the option of completing the task that holds the lock then retrying, or skipping the files completely.

As with all hard disk back-up programs, the integrity of your data is of primary importance: bells and whistles come second. XCO allows you to generate a 'listing file' of all files copied. This file will contain not only the file names but also all of the file attributes and a CRC checksum value for each entry. You

can then use the (supplied) program XCOcrc, from the CLI, to check the integrity of the files XCO has written.

XCO will use up to four disk drives to write the back-up. This is a real time-saver since you can preload the drives. XCO can also use up to four drives to write duplicate back-up copies. In other words, it will write the same data to all four destination drives. This make it a valuable tool even to people who don't have a hard disk. If you need to make multiple copies of a disk quickly, you can set up XCO to use a floppy as the source and write to multiple destination devices. This would be a real time saver for user group librarians, among others

Summary

Which is the best? It all depends on your needs.

All of the programs claim to be fast. Fast is relative. With verify turned off, Quarterback and XCO will write a disk in less than a minute. Performing a hard disk back-up without verifying the writes, in my opinion, is a waste of time - you would not be doing the back-up at all unless your data were important and not easily recreated. When you use the verify option, the back-up time increases considerably. XCO and Quarterback are about even in speed. XCO is faster, but writes less data per disk. It evens out over the entire back-up process. saf-T-net, though it writes more data per disk, is considerably slower.

If you need to back-up to/from non-trackdisk.devices you need Quarterback. If you need the back-up disks to be in AmigaDOS format, XCO is the choice. If you want to utilize the most capacity on your back-up disks with the added security of placing the file catalogue anywhere you want, it's saf-T-net. Whatever you decide, don't find yourself with a hard drive that suddenly crashed without a back-up!

For my timing tests, I used a directory that contained an equal number of files 5-50K, 50-100K and 100-300K in length. Testing was done on an A2000 with an A2090 and a MiniScribe 8425F (40 ms) hard drive.

	Q'back	saf-T-net	XCO
AmigaDOS Format	NO	NO	YES
'No verify' option	YES	NO	YES
Non-trackdisk support	YES	NO	NO
Report Generation	YES	YES	YES
Backup files larger than disk	YES	YES	YES
Write a disk - No Verify	57 secs	n/a	48 secs
Write a disk - Verify	1:46	2:21	1:28
FFS floppy support	NO	NO	YES
Multiple floppy support	YES	NO	YES
Multiple copy support	NO	NO	YES
Format disks while running	YES	YES	YES
Runs in 512k	YES	YES	YES
Copy protected	NO	NO	NO
Suggested Retail Price	$69.95	$49.95	$44.95

The Disk Mechanic

Give your disks a Tune-up

Only $89.95

The Disk Mechanic is a comprehensive collection of utilities for Amiga DOS, including TuneUp, DoubleBack, DiskRepair, and WorkShop.

TuneUp: With TuneUp you can improve hard disk or floppy disk performance by up to 400%. Not a caching program, TuneUp is a sophisticated disk optimizer that actually re-organizes the information on your disk to improve system efficiency. TuneUp is fully compatible with Amiga DOS.

DoubleBack: DoubleBack is a high speed archival hard disk back-up program. DoubleBack supports multiple source and destination drives, file selection by name, date, protection, directory, and file note. Back up a full 20 megabytes in under 40 minutes. Selectively copy files between hard disks and partitions. Protect your data!

Sample TuneUp benchmark of some standard 3.5" disks:

Volume Name	TuneUp	BEFORE	AFTER	IMPROVED
Workbench 1.3	3:30	4.86	2.70	79%
Digi!	3:45	9.44	5.22	80%
WordPerfect	4:42	12.36	5.37	130%
SuperBase	3:37	19.39	9.66	100%
data	1:43	14.30	4.68	205%
archives	3:48	28.43	9.52	299%

* Times given are the average of three runs on a standard A2000 internal floppy drive. The TuneUp time is the time required to process the disk (in minutes and seconds). The BEFORE and AFTER times are the time (in seconds) required to get a directory of the disk. The more use a disk has had, the better the improvement!

Demo disk Available; call for details.

DiskRepair: With DiskRepair it is possible to recover accidently deleted or discarded files, salvage files from damaged disks, and reconstruct a corrupted hard disk. DiskRepair works with hard and floppy disks, and is compatible with the Fast Filing System.

WorkShop: For the expert user, the disk WorkShop is a full featured disk file and block editor. Examine and modify disk files, search for viruses, and change menus or requesters in popular programs.

The Disk Mechanic also includes a dozen small CLI based utilities to make managing your disks easier. We are now shipping version 2.5 with full support for Amiga DOS 1.3 and the Fast Filing System. The Disk Mechanic is not copy protected.

Dealer inquiries welcome.
Amiga is a trademark of Commodore Amiga Inc.
WordPerfect is a trademark of WordPerfect Corp.
Digi! is a trademark of Aegis Development
SuperBase is a trademark of Precision Software, Ltd.

Lake Forest Logic Inc.
28101 E Ballard Road
Lake Forest, IL 60045
(312) 816-6666

MasterCard VISA

MCP Associates Inc. *The DISK SPECIALISTS* DISTRIBUTORS

Featuring C.Itoh, Precision, Sony, Etc:
3.5", 100% Certified, Lifetime Guarantee

Bulk Disks
Prices per disk Min: 100 Disks C.ITOH

SSDD—Blue	DSDD—Blue	DSDD—Rainbow
.95	1.19	1.29

Sony, Precision, other brands available.
Call/Write for Disk Pricing & Amiga Product Listings.
Prices subject to change without notice.
S/H—$5.50 min.; COD's add $3.00.
Educational, Corporate & Dealer inquiries invited.
P.O. Accepted,
WINTER SALE SPECIAL! Limited Time Offer!

MCP Associates, Inc.,
PO Box 6260, Dept TA, L.I.C., NY 11106-1933
Tel: (718) 956-9000 Fax: (718) 956-9028
Exceptional Service & Quality

DR. OXIDE SLICES PRICES!

Comp-U-Save's cost-cutting clinician is at it again! This month Dr. Oxide, a bargain-basement surgeon if ever there was one, offers the industry's lowest prices on hardware and software, plus special deals on products available only through Comp-U-Save!

BUILD YOUR OWN HARD DRIVE KITS!

5¼" powered hard drive chassis with fan
3½" powered hard drive chassis with fan
only $120.00 each

1000 or 500 SCSI interfaces with passthru and software for
$145.00

All you need is a SCSI hard drive and you're ready to Rock and Roll!

Call for SCSI hard drive prices

AMIGA PUBLIC DOMAIN

OVER 800 DISKS!
Largest Amiga PD Library in the World!
also C-64 & C-128 — Amiga PD — $4.00 each
Write for Free Catalogue

EXTERNAL SLOT EXPANSION FOR AMIGA 500 and 1000

Now you can use most of the expansion cards designed for the 2000-hard disk controllers, 2/4/8 meg RAM cards, A2088 Bridgeboard, etc. Use low cost IBM compatible expansion cards already supporting a wide range of business and scientific applications. Designed to work with auto configurable cards. Meets Zorro Bus and Amiga 2000 Bus electrical specifications. Available exclusively through Comp-U-Save!

"THE INTERN 500" **2 — A2000 slots — $179.00**
"THE INTERN 1000" **2 — A2000 slots — $179.00**
with power $225.00

DR. OXIDE INTRODUCES *"THE SURGEON"*!

8 slots for your A500 or A1000!

4 — A2000 slots
3 — AT slots
1 — XT slots
With power — case and one drive bay.
Will work with new AT bridge board
$585.00

Amiga Hard Drives 500 - 1000 - 2000

20 Meg—$585.00 32 Meg—$699.99 48 Meg—$799.99

Amiga External Drive $139.99

Only Uses Half the Power of 1010 — with Pass thru

Panasonic WV 1410 Video Camera

For Digitizers—$199.99 16 MM Lens—$29.99

Special 2400 Baud Modem—$149.99

★ Trackball	$45.00
★ Plastic Diskbank (Holds 120 3.5 in. Disks)	$14.99
★ Copy Arm (Heavy Duty)	$29.99
★ Mouse Mat (Fabric)	$ 5.00
★ Mouse Mat (Teflon)	$11.00
★ Gender Changers — All Types	Call
★ Static Mat (23.5 x 25.5 in.)	$24.00
★ Rapid Fire Joysticks	$12.00
★ Printer Buffer (32K-512K)	Call
★ RF Modulator (A1000)	$14.99
★ A/B Switch (Ser.)	$13.99
★ A/B Switch (Par.)	$14.99
★ A/B/C/D/ Switch	$29.99
★ Crossover Box	$39.99
★ Cables 500 — 1000 — 2000	Call
★ Teak Diskbank (holds 150 3.5 in. disks)	$39.99
★ Teak Diskbank (holds 200 5.25 in. disks)	$39.99
★ The Library (holds 80 3.5 in. disks)	$19.99
★ Floppy Wallets (Many Sizes)	Call
★ 3.5 in DS/DD Disks (Bulk)	$1.25 ea.
★ 5.25 in. DS/DD Disks (Bulk)	$.39 ea.
★ Books — All Titles 20% Off	Call
★ Memory 512K — 8 Megs	Call

★ We carry tons of hardware and software for the Amiga. Call for prices.

We welcome dealer inquiries on all our products.

Come See Dr. Oxide
In Our Booth At All
World Of Commodore Shows!

Comp-U-Save

410 Maple Avenue, Westbury, NY 11590

In NY State (516) 997-6707 (Tech Support) Outside NY State (800) 356-9997 (Orders Only) FAX (516) 334-3091

Fonts, Fonts, InterFonts

A tool for creating and converting ray-traced fonts

Review by Jay Gross

Interfont
Ray-traced font utility
Price: $119.95 US
Syndesis, 20 West Street,
Wilmington, Massachusetts 01887. (617) 657-5585.

Have you noticed anything about the Amiga's fonts? I mean their un-expandability. If you stretch an Amiga font, you get big blocky letters. Round portions of the letters will look particularly uncool. Oh, and another thing. The Amiga's screen fonts are either small in screen size or *huge* in disk space. Moreover, you're left holding the Bagatelle-Medium-Italic if you want to use them in a ray tracing program. A certain other computer on the market touts its 'scalable' fonts - enlargeable to the max. Not so on the Amiga. A 9-point font will look a mess if you stretch it to 99 points.

Suppose, though, that you made Amiga fonts in the same object-oriented way you make objects for your ray tracing software (you *do* trace rays, don't you?). Nice, *scalable* letters made up of a whole bunch of polygons that the ray tracer can happily trace. You could stretch 'em, squish 'em, extrude 'em or anything else, no problem. No jaggy, stair-steppy circles and diagonals. You could put these fonts into a ray-tracing program and have at them like any other object, or squash and stretch them all you want in one of the desktop publishing packages.

For doing just that, the Amiga software shelves already sport a few structured fonts collections. In addition, some Amiga CAD and video titling software packages offer object-oriented letters, and at least one ray tracer comes with a couple of sets of letter objects.

A new package

There's a new package on the block, though. It's InterFont, from Syndesis, and it's a piece of software for *creating* object-oriented fonts as well as readily converting them among any of the several ray-tracers' file formats, as well as Aegis' Draw Plus CAD format. See, since all the file formats for all of the ray-tracers aren't even vaguely the same, object-oriented fonts aren't nearly as universal to object programs as the Amiga's screen fonts are to bitmap graphics programs - unless you can convert 'em.

InterFont's sister program, InterChange, fixes that problem, too. InterFont is actually a 'module' of InterChange, which is Syndesis' program to make one ray-tracer's objects palatable to other ray-trace programs and the like. For example, InterChange chows down on a Turbo Silver object and spits out a Sculpt-Animate 3D (or 4D) object. It consumes a Videoscape 3D object and concocts one that Aegis' Draw Plus won't choke on. Yes folks, InterChange is the United Nations Delicatessen of structured object formats.

Two main parts

The InterFont package has two main pieces, plus InterChange and its basic modules (as well as a few example fonts already made up, and a bunch of nice pictures). The InterFont Designer program lets the user create structured fonts, which it saves in its own file format. The easy way to get a structured font is to trace a bitmapped one with structured graphics. (Well, the *really* easy way is to use the ones that are supplied with the program, but that's the *lazy* way, too!).

To get a whole, structured font, you have to trace a whole, unstructured character set, and that takes a while to do. Count up the objects - 26 lowercase, 26 capitals, ten numerals, plus punctuation, accented characters... - and you'll see why. However, you can trace only the letters you need, if you wish. You don't even have to make letter shapes - anything you can define with polygons is fair game. You can then type them at the keyboard to generate individual or combination objects for your ray-tracing pleasure. Don't forget which key does what.

You also don't *have* to trace bitmapped fonts. You can do them from scratch in a blank window if you like. Tracing them is easier! InterFont's drawing tools for this purpose are pretty neat. The best one for tracing purposes is 'Trace', of course. The program's spline tool makes nice looking curves, but when the object converter gets hold of the font, it usually has to convert the curves to triangles (to make them palatable to

ray-tracers and the like), so there isn't much use in making them curves in the first place. Besides, the splines slow everything down considerably - including the InterFont Designer, the InterFont object maker module, and especially any ray-tracing or rendering program that honours splines.

The InterFont Designer program works in colours, but of course the objects are just plain ol' vectors. The colours are useful for easy visualizing during the creative process, however. No, the colour isn't really necessary, but this *is* an Amiga, so live a little. While you work, the program provides a window to show you what your character object will look like, and a reference character (if you're tracing a bitmap). A pop-up window keeps track of the font statistics, such as the number of points in your font (the fewer of these you use, the faster your ray-tracing and object generation will go).

Doughnuts and holes

A special 'hole' colour is provided for making holes in objects - like A, D, O, etc. Since few ray-tracing programs support objects with holes, InterFont's conversion module goes to a great deal of trouble - a clever trick, indeed! - to construct an object with vectored faces that define the hole.

For example, say you're rolling dough. That's 'rolling', not 'rolling in'. Okay, you're rolling dough, and you're going to make doughnuts. To get the holes in the doughnuts, you could stamp them out of the circles of dough, which would make them 'objects containing holes' - or you could make up the doughnut shape by rolling a length of cylindrical dough and joining it end-to-end. That would create the well-known 'torus' shape.

To get holes in its letter objects that a ray-tracer can trace rays *through* - even if it doesn't support 'objects containing holes', InterFont goes the extra mile. It vectors all the holes so the ray-tracer will think it's a complex shape and trace all those faces inside the holes.

Of course, if you really want objects with flat faces and white polygons where the holes go, InterFont does that too. That option is particularly useful when moving objects to Draw format, for use in Professional Page. If you make the holes transparent, anything underneath the middles of the O's, P's, D's, and such will show. Making them white prevents stuff underneath from shining through.

Although the tools provided in InterFont Designer are quite sufficient to accomplish the task, it'd be nice to have a few more drawing tools, particularly a square and a circle - a circle being the most urgent. Circles are hard to draw, even when tracing from a bitmapped font. The program would also be a bit more convenient to operate if there were a few more on-screen buttons, replacing the pulldown menu options. It'd be nice to be able to manipulate a font's peculiarities out of the program, too - things like character width and height, baseline, kerning, etc. Well, you know how it is... no matter how much you give 'em in a software package, they always want more. InterFont gives a lot, and the incredible complexity of what the program is doing is hidden from the user behind a very intuitive user interface.

The task of making the font in the first place - and again, you're not limited to drawing letters, but the program will save your objects as though you had done so - is the time-consuming part of InterFont's operation, especially if you choose to trace an entire character set. The next step is to make objects that your target program will be able to deal with. You do that with InterChange (which comes with the program in case you don't already have it) and its InterFont module. Crank up an InterChange module for each of the file formats for which you want objects. These can include VideoScape, Sculpt, and Draw (all of which come with the program), or Forms in Flight and Turbo Silver 2.0 and 3.0 (with optional modules), or Professional Draw (with optional Real Soon Now module).

The InterChange environment is a study in interprocess communication. Each of the separate modules - all runnable from the Amiga Workbench - has its own little window, some with selectable options. InterFont has a conversion module with a screenful of options, and InterChange itself manages all of the above with file requesters, status window, and path specifiers. It's a sight to behold. There is undoubtedly an easier way to do it, but this way is very 'modular' and easily expandable as (inevitably) more new structured object formats are invented and marketed.

For example, Syndesis is beta-testing a new module that will add Gold Disk's new Professional Draw to InterFont's (and InterChange's) object format capabilities. In addition, since the file format for Turbo Silver 2.0 and 3.0 are slightly different, another new module in beta-test will add those conversions to the list. Syndesis' conversion modules are $19.95 list.

After you get all those programs running, creating an object consisting of a bunch of letters is an easy task. The InterFont conversion module springs to the front as its turn comes by (all this is managed by InterChange), asking for the strings you want. Your object can have multiple lines, multiple words, and any attributes you could apply to regular text, including underline, bold, and italics.

After that, InterChange purrs for a while, and presto! An object for your ray tracer or Professional Page (which imports Aegis' Draw Plus format) or whatever. It's just like magic, only with less work.

Points to ponder

To squeeze the letters into some of the Amiga programs that have limits on the number of points an object can contain (Videoscape, for example), InterChange includes a 'Point Reduction Tool'. This clever module figures out where on an object there are common points and makes them into one

point. Even on a huge, inefficiently drawn object, the point reduction process takes only a few minutes.

Throughout the InterChange and InterFont creation process, the program announces what it's doing and what it's planning to do. These messages are displayed more for reassurance than information, since there isn't much you can do to stop the process most of the time. The program's impressive, graphic STOP button is non-functional except during pauses in the processes, which aren't necessarily where you'd like to stop things - like just after you've inadvertently started something going that takes a long time to finish.

With this and only a few other minor problems to complain about, InterFont passes all the reviewer's main tests with honours. It multitasks happily with other programs - though the screen does get busy with all those windows going. Its user interface is intuitive and informative. It's easily (and nicely) operable from the Workbench - Syndesis has gone to a great deal of trouble on this one. Its documentation is adequate - although there are a few things that bear some rummaging. And it's productive. *Very* productive. Those packages of ray-traceable fonts are going for $79, and you get only a few. For InterFont's $119.95 US you get several fonts as well as a nice tool for making as many as you want and a conversion program for porting objects around - even the ones you buy from some other company.

DEBUG Version 2.1

Attention Benchmark Modula-2,™ CAPE 68K,™ Lattice 'C'™ programmers.

Now, catching your bugs is easier with LDebug.

- View your source as it executes.
- Trace by source line or MC68000 instruction.
- Supports multiple source modules.
- Set breakpoints by clicking on a line of source code.
- Global variable watch window.
- Disassemble a source line by double clicking on it.

Ask your dealer for LDebug, or send check or money order for $49.50 U.S. to:

LogiComp Software
4010 Woods Blvd.
Lincoln, NE 68502
Phone: (402) 423-6768

Benchmark Modula-2, trademark of AvantGarde Software, CAPE 68K, trademark of Inovatronics, Inc., Lattice 'C', trademark of Lattice, Inc.

ARexx

The REXX Language for the Amiga

ARexx is a multitasking implementation of the REXX language, an elegant high-level language especially suited for macro-processing. Its clean, simple syntax makes REXX easy to learn for novices — and experienced programmers will appreciate advanced features like compound variables, INTERPRET instruction, and source-level debugging.

- Interactive, Interpreted Operation
- Exceptional String-handling Facilities
- Built-In Library with Over 75 Functions
- Built-In Source-Level Debugger
- Compact, Reentrant code — Only 32K

The Emerging Standard

ARexx is the *de facto* standard for inter-program communication. Software that supports the ARexx command interface can be customized, extended, and combined with other ARexx-compatible software in integrated applications. Watch for announcements of ARexx support from a growing list of Amiga software vendors!

Available Now ... Only $49.95

WShell

The Command Shell You've Always Wanted

WShell is the best command shell on the market, with features and convenience of use that go well beyond the CLI and 1.3 AmigaShell. And yet it's highly compatible with the CLI, so you don't have to learn a new command language to use it!

- Line Editing/Command History/Aliases
- Built-In Commands/Resident Commands
- Prompt String/Window Title "Variables"
- Concurrent Piping
- Transparent Support for ARexx Macros
- Script-bit Support for "Execute" Scripts
- Compact, Reentrant Code — Only 10K

ARexx Command Interface

With WShell you can use REXX-language macro programs like executables — even as filter programs in a piping system. Use the source-level debugging to single-step through a system of concurrently-running "piped" programs!

Available Now ... Only $50

Ask your Amiga dealer for ARexx and WShell, or order them directly from the author. Please include a check or money order plus $2/item shipping ($8 for overseas airmail) and any applicable taxes. Dealer inquiries welcome!

William S. Hawes
P.O. Box 308
Maynard, MA 01754
(508) 568-8695

Amiga is a trademark of Commodore-Amiga, Inc.

DevPac Amiga

*A **fast** 68000 assembler development system*

Review by Nick Sullivan

***DevPac Amiga, Version 2,** by HiSoft*
Distributed by MichTron Inc., 576 South Telegraph, Pontiac, MI 48053, (313) 334-5700.
Suggested retail price $99.95 US

The HiSoft company of Britain are developing impressive language tools for the Amiga and other 68000 machines, including the BASIC compiler reviewed elsewhere in this issue. Their DevPac assembler package, based on a similar product for the Atari ST, is both full-featured and almost unbelievably *fast*.

Besides the assembler itself, DevPac includes an adequate though unspectacular text editor, a symbolic debugger, two sets of include files, and some demo source. The main component programs go by the awkward-sounding names *GenIm2* (the assembler), *GenAm2* (the editor), and *MonAm2* (the debugger). The PD version of Software Distillery's *Blink* is provided as the standard linker.

GenAm2 (the source editor)

We'll start here because, like an increasing number of other language packages these days, DevPac can function as an integrated system that lets you invoke the assembler and the debugger directly from the editor. The integration is handled privately rather than through ARexx, unfortunately, so you *have* to use the *GenAm2*, and not 'your favourite text editor', if you wish to take advantage of it.

GenAm2 - which appears to be fundamentally the same program as the editor provided with HiSoft's BASIC compiler - does have virtues of its own: it is solid and quick, and - like all the programs in this set - very compact (less than 24 kilobytes). It is not nearly as feature-laden, however, as Uedit, CEDPro, and other popular programmers' editors for the Amiga. Unlike those programs, too, it is driven mostly by Wordstar-based commands (CTRL-key combinations) for its editing commands, perhaps not the ideal model for an Amiga text editor.

There are other signs of a hasty and somewhat inadequate port to the Amiga: one is the use of the ARP library file requester, with its peculiar scroll gadget; another is the inconsistent and non-standard placement of the OK and CANCEL gadgets in the other requesters. Only one file can be edited at a time, and that file occupies a fixed-size, contiguous buffer. All in all, this editor is no jewel even by PD standards: unless you really can't live without the integrated environment, you're not likely to get much use from it.

And the integration does not really go very far, though there are some clever touches. The assembler can generate either linkable or stand-alone code, and can be instructed to send its output either to memory - for immediate testing - or to a disk file. You can also have it generate no output at all, which is to say you are using the assembler as a syntax checker.

Since there is no provision for linking from within the editor, however, immediate testing is only possible for programs that are assembled from a single source file, and even then, there is no way to pass a command line to the program being tested. These limitations combine to prevent many, and perhaps, most programs from partaking in the rapid development cycle *GenAm2* was designed to provide.

GenIm2 - (the assembler)

This is the centrepiece of the DevPac suite, and probably the only good reason for purchasing the package. But it *is* a good reason.

One assembly language file I've been working on lately has grown somewhat out of control, to 50 kilobytes (including fairly abundant commentary) of source - just over 1500 lines. Running from the editor, and generating the object code to RAM:, *GenIm2* processes this file in something like four seconds! To one who has learned patience the hard way - from C compilers - this kind of alacrity on the part of a language translator seems little short of miraculous, even when one takes into account that an assembler's task is very much simpler than that of a high level language compiler.

I didn't run any timing tests more sophisticated than the one just described, so I can't give you 'independent laboratory test results'. From what I have seen, though, I have no reason to doubt HiSoft's claim of 35,000 lines per minute for the assembler's minimum cruising speed, with burst rates of up to 70,000 lines per minute.

Nor does this spectacular performance depend on cutting other corners in the program specification. *GenIm2* supports virtually every option of the MetaComco 'standard' Amiga assembler, plus various switchable optimizations (such as converting backward branches to short where possible), and enhancements (such as allowing a more readable format for local labels). The 68010 and 68020 CPUs, and the 6888x math chips are *not* supported, however. Source intended for the MetaComco or other standard assemblers can be handled by *GenIm2* with very little or no modification.

MonAm2 (the debugger)

Though less restricted than the text editor, *MonAm2* is again not as full-featured as other programs available to the Amiga programmer. Still, it again packs a surprising amount of functionality into a very compact space. All the usual debugger functions *are* supported, including symbolic disassemblies, memory and register displays and modification in various flavours, memory searching, breakpoints simple and conditional, and single-stepping through program execution.

You cannot debug multiple tasks simultaneously, nor sub-tasks spawned from within a program. There is no support for macros in any form, and the user interface is, to put it kindly, lousy; the HiSoft programmers don't seem to excel in that department. The program is entirely keyboard-driven, and strangely enough leans heavily on Right-Amiga key sequences - menu shortcuts without the menus. The manual states that the commands are virtually identical to those used by the pre-existing Atari ST version of the program; one suspects that the resemblance in fact goes a great deal deeper. *MonAm2* maintains several window-like displays (giving it a feel something like MetaScope) on a custom screen that really has only one backdrop window; probably this arrangement reflects its origin on a single-tasking, single-display computer.

There are some nice features. The register display shows not only the contents of each register, but also the contents of the first three long-words of memory the register points to, which, for address registers at least, makes a lot of sense. The search command allows you to hunt through memory for bytes, words, long-words and ASCII strings, as you'd expect, but it also let you search for machine language instructions and operands expressed in mnemonic form. For example, a search for "$14(A" will match an instruction like "MOVE.L D2,$14(A0)", to borrow an example from the manual. As far as I know, this particular feature is not to be found in other debuggers currently available for the Amiga.

The manual and sundries

DevPac's 200-page manual (thankfully not called *ManAm2*) is in most ways adequate, providing detailed reference information and a minimal tutorial on each of the component programs except Blink, which is covered only very briefly. The manual also offers more of the same kinds of thoughtful features that are sprinkled erratically throughout this package, such as a complete listing of the 68000 instruction set, a synopsis of the CLI commands, brief discussions of the Amiga libraries and include files, and of the 68000 CPU exceptions, and suggestions for further reading. The concept of proofreading is apparently just as foreign to HiSoft's manual writers as to most other software documenters, but the material *is* clear and complete.

The other major component of the package not discussed so far is the set of include files. Actually, there are two sets: one is in the standard CBM/MetaComco format; the other is a DevPac-specific set that takes advantage of the special *GenIm2* directives RS.B, RS.W, RS.L, RSRESET and RSSET.

These directives provide a more efficient means of declaring data structures than the macros used for the same purpose by Commodore, without loss of compatibility. For instance, here is the standard assembler declaration of the Node data structure, along with its DevPac equivalent (right):

```
STRUCTURE  LN,0                    rsreset
   APTR  LN_SUCC          LN       rs.b  0
   APTR  LN_PRED          LN_SUCC  rs.l  1
   UBYTE LN_TYPE          LN_PRED  rs.l  1
   BYTE  LN_PRI           LN_TYPE  rs.b  1
   APTR  LN_NAME          LN_PRI   rs.b  1
   LABEL LN_SIZE          LN_NAME  rs.l  1
                          LN_SIZE  rs.w  0
```

The interpretation of each of the macros in declarations like this (STRUCTURE, APTR, UBYTE, BYTE and LABEL) can add significantly to assembly times, as Amiga assembler programmers quickly discover when they need to include the lengthy definitions of the Intuition structures.

The DevPac includes have also been stripped of comments. These modifications result in a smaller include directory and considerably faster include file assembly than would otherwise by possible. A utility is also provided that converts include files from the standard format to the DevPac format automatically, so users of the non-standard files can retain compatibility across new releases of the operating system.

Summary

The heart of this package, the assembler, is a very strong program by people who clearly know what they are about. Less than 28K in size, it can withstand feature-by-feature comparison with just about any assembler on the market - except for its lack of support of the post-68000 chips - and its speed can probably not be substantially improved upon.

The other components are not less than adequate, and the debugger in particular has a lot of merit. It's too bad that this package has not benefited more from its port to the Amiga environment; perhaps in a future release it will have been adapted to the machine more thoroughly. Even so, it is a good product, and recommended.

HiSoft BASIC Professional

Too good to be BASIC

Review by Chris Zamara

Integrated text editor/BASIC compiler
Distributed by Michtron
576 S. Telegraph, Pontiac, MI 48053
Suggested Retail: $159.95 US

"A compiler for AmigaBasic - ho hum". That may be the typical reaction of programmers who struggled with the less-than-polished implementation that is still shipped with every Amiga. And while BASIC is often useful for testing an algorithm or quickly prototyping a design, very few software developers would consider writing an actual application with it.

With *Hisoft Basic Professional,* (*HBP*) those attitudes are seriously challenged. The compilation process is made painless by a fast, to-the-point text editor that is integrated with the compiler: a single key combination compiles and runs the current program. Compile times are very good - good enough that short programs can be developed with almost the same immediate feedback afforded by an interpreter.

But program development is, of course, the area where a compiler is *usually* inferior to an interpreter; it is the creation of stand-alone executable programs that makes a compiled language a development tool for real applications. In this regard, *HBP* does not disappoint: the programs it creates are surprisingly compact, especially considering the amount of baggage that accompanies a BASIC environment. They are also surprisingly fast. You have the option of producing programs that will work with a special shared library that you put in your 'libs:' directory, or a completely stand-alone program. The former option will produce a more compact program, but the latter allows you to create a program that you can distribute to any user - and one that in no way gives away its BASIC origins.

But what of the language itself? While *HBP* will compile AmigaBasic programs, it is by no means an AmigaBasic clone - it is actually a superset of that language. The manual claims that it will compile Microsoft QuickBASIC programs with minimal changes, and that it will compile most flavours of BASIC with little modification! We can't verify these claims conclusively, but our unscientific testing methods back them up: type just about any code that comes to mind (a mind tainted by BASICs that include the Commodore PET and 64 variety) and it will usually compile and do what you'd expect. That is probably the most surprising - and useful - characteristic of this system. Its chameleon-like qualities will endear it to programmers who just want to enter Do-What-I-Mean pseudo-code and get the job done. It also means that you can easily port BASIC programs from other systems, which can be a good first step to properly Amigacizing a program (sorry, folks, direct ports just don't cut it!).

If you're not interested in a universal BASIC dialect, but just want a replacement for AmigaBasic, *HBP* adds some worthwhile enhancements. Perhaps the most significant of these is recursive sub-procedures and functions, sorely missing from AmigaBasic.

The compiler has a number of options to turn off various checks for overflow, array subscript checking, etc. Once your program is thoroughly debugged, you can turn off the checking of your choice to make the final program smaller and faster. There is an option for stack checking, but there seems to be a bug in that area: it's possible to crash the machine by writing a function that calls itself indefinitely - an easy mistake to make when writing recursive routines.

To give an example of the size of programs that are generated, we wrote a simple test program using a custom screen, a custom colour palette, a window, some menu options, and some menu-driven graphics effects. Using the shared library option, the executable program came out to only around 1500 bytes! The shared library is over 46k, so that's not as tiny as it seems, but remember, the shared library only exists once, no matter how many Hisoft-generated programs you have. This is a great way to save disk and memory space when using many small programs. Turning off all the checking options, the test program was just 1200 bytes. Unfortunately, we were unable to create a stand-alone version of the program (one not requiring the Hisoft shared library) because of a problem with the particular package we received: the package consists of a two-disk set, but we found that both disks in the package, while labelled properly, were actually identical. So identical, in fact, that AmigaDOS choked when they were both inserted in separate drives. Fortunately, disk 1 - the one we have - contains the compiler, but the larger version of the compiler that is capable

of generating stand-alone programs is on disk 2. We don't know if our package is a fluke, or if others were shipped with the same problem, but we had to do without stand-alone program testing for this review.

Instead of generating executable programs, you can create object files that you can link with other object files created from C or assembler source; you can call these C or assembler functions from your BASIC programs. If this is sounding more like a real compiled language and less like the plaything that most BASICs are, you're beginning to get the point.

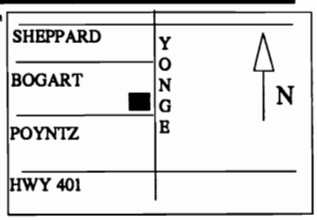
The durable, 358-page coil-bound manual meets the high standards set by the software. The compiler system, examples and programming concepts are covered in the first few chapters, and the remainder is a reference guide (in alphabetical order) to all of the language's statements and functions.

Overall, the implementation of this package is superb, and, from a programmer's point of view, it's one of the most interesting products that we've seen for the Amiga. If you're a beginning or casual programmer, *HBP* will let you create efficient and useful programs that you might not have been able to create otherwise. If you're already an Amiga programmer, your choices for application development languages may have just grown beyond the traditional Assembler, C, and Modula-2 alternatives.

CANADA'S BEST SELECTION OF
AMIGA - C64 - C128 SOFTWARE
COMSPEC HARD DRIVES AND BERNOULLI DRIVES
AVAILABLE FOR
THE AMIGA 500 AND 2000

Electronics 2001
COMPUTER PRODUCT CENTRE

4736 YONGE STREET
WILLOWDALE ,ONT
M2N 5M6
TEL: (416) 229-2700



AUTHORIZED COMMODORE DEALER SINCE 1979

New Store!
Fisher's
Computers & Software
Authorized Amiga Dealer
8005 Archibald
Rancho Cucamonga, CA 91730
(714) 987-1662
Serving The Inland Impire

The fastest
Modula-2
Software Development System for
AMIGA
$ 249
Demonstration disk $ 10

Modula-2 is the successor of Pascal. Both languages were invented by Professor Niklaus Wirth. Modula-2 is an all purpose language suited for educational, home and professional use. Modula-2 leads to good programming technique, easy readability and maintainable programs. In fact, a lot of schools and universities have chosen Modula-2 for their programming courses.
M2Amiga is a software development system for Modula-2 based on an extremly fast single pass compiler. It is fully integrated into Amiga Workbench and is easy to use from CLI or Workbench. It runs on all Amiga models with a minimum configuration of 512k RAM and one disk drive.
M2Amiga has been developed specially for the Amiga and therefore optimally supports the features of this unique computer.
- Single pass compiler produces optimized native MC68000 code from standard ASCII files. Full support of latest edition of Modula-2.
- Fastest library calls possible on the Amiga. No intermediate code needed for library calls and parameter passing.
- Supports FFP, 32 bit IEEE and 64 bit IEEE reals formats, you can even mix all three formats within the same program.
- Easy access to MC68000 registers, allows the insertion of inline code.
- Full interface definition to the Amiga's operating system following the official definitions.
- The best runtime system currently available on the Amiga. Opens and closes all used libraries, captures all GURUs and releases all used resources after program crash or termination. Routines for System requesters enable user interface consistent with the Amiga's Intuition.
- The emacs based editor places the cursor on compilation errors and explains them in full text.
- In just a few seconds the linker produces stand-alone applications that can be run from CLI or Workbench. Stand-alone applications do not contain any useless overhead – they are compact and fast.
- All programs can produce icons for their output files to allow effective use of the intuitive Workbench environment. Experience the new feeling of program development by selecting and double clicking icons!
- Also included are a project management tool and other utilities.
- Besides the handy manual, an introductory guide to Modula-2 is included. This documentation allows both the beginner and the advanced user to find any information needed quickly and easily.
- Demo programs show the use of various features of the Amiga.
- For professional program development special utilities are available.
  + Source Level Debugger, the new way of debugging your programs. Inspect all your variables (displayed in their own format), zoom into structures as records or arrays or follow pointer chains by clicking the mouse buttons
  + Object file converter, converts standard Amiga object files into M2Amiga format. This allows linking of routines written in other languages.
  + Linker for devices or libraries. Add power to the Amiga by supplying new libraries and devices. Easy and intuitive programming instead of fooling around with Assembler and Amiga linker are possible now!
  + Modula-2 Amiga Programming System Environment

We have Modula-2 Compilers for HP-UX, IBM/370, PCs (Taylor, M2SDS, JPI), OS-9 and Sun, tomorrow even more!
Texas residents add 8% sales tax. International Orders add $ 20 shipping.

The Modula-2 people:
INTERFACE TECHNOLOGIES
3336 Richmond, Suite 323, Houston, TX 77098-9990 (713) 523 8422
Dealer inquiries welcome 1-800-922-9049

2U
A. + L. Meier-Vogt
Im Späten 23
CH-8906 Bonstetten/ZH
Switzerland
Tel. (41) (1) 700 30 37

News LINK

BADGE Killer Demo contest winners: The Bay area Amiga Developers GroupE has announced the winners of the 1988 Killer Demo contest. The primary purpose of the contest is to create demo disks for dealers to use to show off the Amiga.

The first prize winner in the tool-based demos division was Brad Schenck for his *Director*-based animation *Charon*, followed by *Not Boing Again* by Dr. Gandalf and *Tychoid* by John M. Olsen. Best custom-programmed demo was *Tank* by Vince Lee, followed by *Triple,* by AmigaTeX author Tom Rokicki, and *Multitasking,* by author/developer Rob Peck. Brad Schenck also won Best Overall prize (an Amiga 2000 donated by Paul Higginbottom of CBM) and Best Sound. John Olsen also won Best Graphics for *Tychoid* and Dr. Gandalf's *Not Boing Again* won in the Funniest category.

Two disks with the top three demos are available from BADGE for $5.00 US by sending a request to BADGE Killer Demo Contest, c/o Randy Spencer, P.O. Box 4542, Berkeley, CA 94704, USA. All of the entries will be sent to Fred Fish for distribution in his Freely Redistributable Library.

The third BADGE Killer Demo contest will be held at the September, 1989 BADGE meeting. Demos will be accepted up to and including September 1, 1989. For a free copy of the rules, send a self-addressed, business-sized envelope to the address below. Only serious enquiries will be considered. Contact: BADGE Contest, c/o DATAPATH, P.O. Box 31008, Los Gatos, CA 95031, USA.

Software synchronizes graphics and audio: Mindware International has announced *PageSync*, a tool for creating timed interaction between Amiga-based graphics and audio. It allows external MIDI events and computer animation to control each other.

Used in conjunction with a MIDI instrument, *PageSync* will let a computer animation advance, stop or reverse, based upon signals generated by that MIDI instrument. For example, a MIDI drum machine could cause a cartoon to advance by one frame per drum beat, keeping the animation in perfect time with the music. The program will also do the opposite: an animation playing on the Amiga could literally play a MIDI instrument while running the animation. *PageSync* will also synchronize computer graphic events and digitized sound samples.

PageSync will also make it possible to integrate computer graphics and music with video, through the use of additional hardware. For example, users could make their own home music video by sending synchronized graphics and music to a standard video cassette recorder. In addition, live images from a video camera could be merged with graphics into a single image through the use of a genlock. This image, along with the synchronized music, could then be directed to videotape. The program carries a retail price of $100 (US), and will work in conjunction with two other Mindware products: *PageFlipper Plus F/X* and *PAGErender 3D*. It is suitable for such applications as music, corporate sales presentations, training, animation and home video. For more information, contact Mindware International, 33 Alliance Blvd., Unit 1, Barrie, ON, L4M 5K2, Canada. Telephone (705) 717-5998.

Technical information from CBM: The latest list of technical information available from CATS includes the *AmigaDOS V1.3 Native Developer Update*. This four-disk set consists of:

- Include 1.3, the commented C and assembler include files
- Include.strip 1.3, which provides stripped include files, 1.3 startups, *Alink, Amiga.lib, debug.lib, ddebug.lib, atom,* LVO offset lists and more
- AutoDocs 1.3, which provides autodocs that summarize library, resource and device calls
- Readme 1.3, which includes sample 1.3 printer driver sources

Cost is $20.00 US, $22.50 US if shipped to Canada.

Also available:

- The *A500/2000 Technical Reference Manual* describes the technical features of the A500 and A2000, as well as those features that differ from the A1000. Table of contents includes: System Block Diagrams, Amiga Expansion, Designing Hardware for the Amiga Expansion Architecture, Driver Documentation, Software for Amiga Expansion, PC Bridgeboard and description of the PC/XT emulator for the A2000, BIOS entry points, Custom Chips, Miscellaneous Hardware Information, Clock/Calendar registers, Power Budgets, A2000 PAL equations and schematics. Cost is $40.00 US.
- The *Software ToolKit* consists of a manual and disk set that includes several utilities to assist developers with Amiga programming: *WACK, Memacs, Debug, Keytoy* and so on. Cost is $25.00 US.
- The *IFF Manual and Disk*, updated October, 1988, consists of a spiral-bound manual that includes full IFF documentation and source listings for Amiga IFF; and a disk that contains source code, object files, executable programs and documentation. Cost is $20.00 US.
- The *1988 Amiga Developers' Conference Notes* is a 400-page manual and disk created for the 1988 Amiga Developers' Conference in Washington, DC. Topics covered include: Unique Amiga Techniques, Tips and Tricks for Programming in C, IFF, Autoboot and Kickstart

V1.3, V1.3 Printer Device and Printer Drivers, Amiga Audio and Sound, Overscan, High Resolution Fonts, A500 Expansion Cards and others. Cost is $75.00 US.

- *Amiga 1000 Schematics and Expansion Specifications* is a spiral-bound manual that contains full Amiga 1000 schematics, timing diagrams, PAL equations and documentation for the autoconfiguration process. Cost is $20.00 US.
- The *Fall 1986 Amiga Developers Conference Notes* contain the diagrams, outlines and additional notes pertaining to each conference speaker's topic. Some additional information has been included, e.g. the 8520 specifications. Some of the topics covered at the conference included: Exec, IFF, System software v1.2, *Amiga Programmers' Suite* by RJ Mical, 3D Graphics/Ray Tracing, Software Expansion Architecture, Hardware, MIDI Sound Synthesis, and Efficient C Programming. Cost is $50.00 US.

Prices listed include shipping charges within the USA. Send checque or money order payable to Commodore Business Machines to: CATS-Orders, 1200 Wilson Drive, West Chester, PA 19380, USA. For orders shipped to Canada, add an additional $2.50 US to the cost of each item. Add $5.00 per item for orders outside of the USA or Canada.

Auto-booting SCSI adaptor for all Amigas: Comspec Communications, which has the distinction of providing the only auto-booting hard drive for the A1000, has extended the line to include auto-booting SCSI interfaces for the A500 and A2000. Like the A1000 product, these provide an interface to such SCSI devices as hard disk drives, Bernoulli Boxes, tape streamers, optical disks, printer servers and others.

The Comspec hard disk driver and Bernoulli Box software allow up to ten hard drives/Bernoulli Systems in sizes from 20 Mb to gigabytes of storage to be connected to the host adaptor. Each drive may be given any three letter name.

Other features include:

- automatic media defect management
- an intelligent SCSI controller chip that leaves the Amiga CPU free for multitasking
- built in clock and bus pass-through on the A1000 interface;
- Macintosh-compatible connector
- autoconfiguration

Comspec says it has recently lowered prices on the Amiga SCSI products. For more information on the SA 500, SA 1000 and SA 2000 and available hard disk drives, contact Comspec Communications, 74 Wingold Ave., Toronto, ON M6B 1P5, Canada.

Compressed A500 internal expansion: Memory And Storage Technology, Inc. has extended its range of miniature peripherals with the MICROMEGS, an alternative to the A501 internal expansion RAM. The company has made the MICROMEGS unit considerably smaller than the A501 by employing 1-megabit DRAMs. This reduces the length of the on-board conductors, minimizing transmission line effects, for greater noise protection. Also, power consumption is lower: the unit requires only 8 mA of current in standby mode, with 20 mA the maximum drawn when it is being continually accessed. The MICROMEGS board also includes a battery-backed real-time clock. For further information, contact M.A.S.T. at 3881 Benatar Way, Chico, CA 95928, USA. Telephone (916) 342-6278.

Light-pen works in overscan: Inkwell Systems, manufacturers of precision light-pen data entry devices and graphics software for Commodore computers, has introduced its Amiga light-pen and driver. Operational in interlace or non-interlace modes, overscan, stand-alone or in conjunction with a two-button mouse and/or the Easyl drawing tablet, the device is compatible with *Deluxe Paint III*.

The light-pen comes with transparent driver software, and features a tri-lobular, lightweight, two-touch switch design. The pen retails for $129.95 US. More information is available from Inkwell Systems, 1050-R Pioneer Way, El Cajon, CA 92020, USA. Telephone (619) 440-7666.

Expert system tool adds *ARexx* support: Version 1.1 of Emerald Intelligence's MAGELLAN expert system tool adds support for *ARexx* in both 'master' and 'slave' modes; an inference loop mode; a real-time clock; improved rule management tools; more meta-logic to control MAGELLAN features from within a knowledge base itself; and PAL video support for the European market. Contact: Emerald Intelligence, 334 S. State St., Ann Arbor, MI 48104, USA.

***Edit Decision List Processor* for *Photon Video*:** MicroIllusions announces "the approach" of the *Edit Decision List Processor* (EDLP) for the *Photon Video* series of desktop video production software. A film or video is usually made up of many bits and pieces taken from several different reels. An edit decision list tells the video editing hardware which piece from which reel goes where and when. Creating such a list can be tedious and expensive when done manually. EDLP enables the user to create the edit decision list off-line. It can then be transfered over the serial port, or via an MS-DOS compatible floppy disk.

Features of EDLP include:

- 999 events per file capacity
- three tracks (one video and two audio) allow for split edits with all three record times displayed on the screen simultaneously
- editing with cut, copy and paste functions
- find command for location of specific events;
- multitasking support
- online help

EDLP works with 30-frame NTSC (drop, non-drop and mixed frame modes), 25 frame EBU (PAL) and 24-frame film modes. The program supports the CMX, ISC, Convergence, United Me-

dia and Grass Valley Group EDL formats. The product will ship at the end of March, 1989. Price is $499.95 US.

Contact: MicroIllusions, 17408 Chatsworth St., Granada Hills, CA 91344, USA.

***PlotPrint* converts structured drawings:** *PlotPrint* from Hi-Tech Graphics transforms vector-oriented pen-plot data files created by CAD/scientific/spreadsheet programs into high-resolution bitmaps that allows any graphics printer to be used instead of an expensive pen plotter.

The program translates all HPGL (Hewlett-Packard Graphics Language) commands, including dialects found in files downloaded from IBM PCs, VAX minis or IBM mainframes. If the graphics printer provides colour output, the user can create separate set-ups for colour, and black and white.

PlotPrint does not use the *Preferences* drivers, which sometimes do not provide maximum dot resolution. Printouts can be of any size up to available memory – postage stamp to printouts extending over several physical pages, narrow or wide carriage.

The program works in two modes: interactive, which lets the user preview plots before printing, and batch mode, which lets the program operate in the background if desired. The program makes full use of Intuition and is provided with a 50-page manual that includes a tutorial on HPGL syntax.

PlotPrint is available in two versions, v1.0 for standard Amigas, and v2.0 for Amigas equipped with the 68020/68881 combination. Cost for the latter is $100 US; for the former, $50 US. Contact: Hi-Tech Graphics, P.O. Box 446, Tallmadge, OH 44278, USA.

Serenade your Amiga: *Harmony* is a hardware/software combination that gives your machine pitch recognition capabilities. After humming the note into the supplied microphone, the computer will place it on a musical staff. Impulse claims 90% accuracy for the product.

Once the first track has been recorded, *Harmony* can be commanded to add notes that harmonize in one of four styles of music: rock and roll, new wave, classical or big band. Alternatively, the user can add up to three tracks with the supplied editor.

Scores created with *Harmony* can be saved in SMUS format. The program will also let the user assign 8SVX instruments to each track.The program will run on all Amiga models and is scheduled for release in April, 1989. Cost will be $199.95 US.

Contact: Impulse, Inc., 6870 Shingle Creek Parkway, #112, Minneapolis, MN 55430, USA. Telephone (612) 566-0221.

More tools from Abacus: Abacus has released two books for Amiga programmers, and a disk of software tools.

Amiga 3-D Graphic Programming in BASIC is said to detail techniques and algorithms for writing three dimensional graphics programs. Users will learn the basics of ray tracing in all resolutions, light sources, shading and more. The book also features: an object editor for creating 3D objects in any resolution; IFF format graphics; many working examples; and an introduction to 3-D math for novices. Cost for the 300-page. book is $19.95 US.

A diskette is also available for the book, at a cost of $14.95 US. It contains AC/BASIC compiled versions of the *EDITOR* and *TRACER* programs, plus complete BASIC source code and a large collection of objects.

More Tips and Tricks for the Amiga, which is volume 12 in Abacus' Amiga Reference Library, contains techniques for using AmigaDOS and WorkBench 1.3 efficiently. Readers can learn how to make their programs more user-friendly using pulldown menus, sliders and tables. They can also learn how to: write a computer virus-checker; disable fast RAM; upgrade to a faster processor; use the NEWCON and PIPE devices; access machine language and C programs from *AmigaBASIC*; and so on. The 224-page book costs $19.95.

The *Amiga DOS Toolbox* package consists of a disk and a copy of the *AmigaDOS Quick Reference Guide*. The disk contains such programs as *DeepCopy*, a disk copier that copies many different formats; *Speeder*, a disk speed-up utility (not a disk cache); and *Diskmon*, a disk editor. Cost is $59.95 US. Contact: Abacus, 5370 52nd Street SE, Grand Rapids, MI 49508, USA. Toll-free order line: (800) 451-4319.

AutoDroid simplifies digitizing: AutoDroid, available from MicroSearch, is a servomotor that attaches to the colour filter wheel used with the *Digi-View* digitizer. It connects to the Amiga's second mouse port. The *Digi-View* software can command the AutoDroid to rotate the filter wheel for each colour scan. Cost is $59.95 US.

MicroSearch has also announced version 2.0 of their desktop publishing program *City Desk*. The program features support of PostScript, Hewlett-Packard's LaserJet+ and *JetSet* HP fonts, 24-pin printers, *WordPerfect*, *Scribble!*, typesetting commands and automatic flowing of text. Contact: MicroSearch, 9896 S.W. Freeway, Houston, TX 77074, USA. Telephone (713) 988-2818.

More colour fonts: Kara Computer Graphics has announced the expansion of their Kara Fonts Headlines Collection to include *Subheads*, a scaled-down version of their *Headlines* series, and *Headlines 2*.

Subheads offers 72- and 55-point font styles in chrome, marble, brick, granite, wood, cast, column, chisel serif, bevel and chisel. The first four of these have been designed with a pixel outline, which can be dragged in any direction to create a wide range of three-dimensional effects.

Headlines 2 presents four three-dimensional colour fonts in two and three sizes each: *chiselSCRIPT,* an italic script in upper and lower case; *Glass,* a transparent gothic face; *Embossed,* a serif-based type style which can appear raised or sunken on a background surface; and *Engraved,* a shiny two-plane gothic type style with a draggable pixel outline. The fonts are high resolution and in eight colours.

Subheads and *Headlines 2* each come in a two disk set, and cost $69.95 apiece. For more information, contact Kara Computer Graphics, Kara Blohm, 6365 Green Valley Circle, No. 317, Culver City, CA 90230, USA. Telephone (213) 670-0493.

Amiga Developers' Reference Guide, Edition 1.3: This spiral-bound manual from Pacific Press is designed as a massive cross reference and index of all 1.3 system functions, macros and symbols. An additional chapter provides a list of all the functions in prototype form, which is useful when the programmer can't remember the argument types and order, return types and the number of arguments. A permuted index is also provided that allows the reader to find a function if he or she knows what it does.

This edition, the third, adds information on ARP, colour coding of sections, assembler specific symbols, and so on. Cost is $19.95 US. For more information, contact Pacific Press, P.O. Box 611075, San Jose, CA 95161-1075, USA. Telephone (408) 741-4250.

French language marketing support from CBM: One third of Canada's population of 26 million claim French as their mother tongue or language of preference. To help reach that market, Commodore is putting in place a French library to be located at the company's Montreal office. Serge Bureau will be responsible for the library and will provide a short critique of French-language products submitted. He will also make them available for demonstrations in French by CBM's territory managers at dealer seminars, educational presentations, consumer shows and government meetings. Interested companies should send ten copies for testing, evaluation and distribution to Serge Bureau, Commodore Business Machines Ltd., 3100 Côte Vertu, Suite 455, St. Laurent, PQ H4R 2J8, Canada.

Call for Amiga art submissions: The ABACI Gallery of Computer Art, in association with Machlan Enterprises, is compiling a book proposal of exclusively computer fine art created on the Amiga.

Artists interested in submitting art should send duplicate slides (no originals accepted), annotated with title, artist's name and copyright notice. All art work should be original, preferably hand-drawn, and created by the artist submitting the work. Deadline for submissions is June 30, 1989.

The art should be sent to: ABACI Gallery of Computer Art, 312 N.W. 10th, Portland, OR 97209, USA. Attention: Daria Barclay. For more information, call Daria at (503) 228-8642 or Richard Machlan at (503) 639-4631.

This ad was laid out in Professional Page from Gold Disk, and printed at this size on the HP DeskJet using PrintScript.

PrintScript

The PostScript interpreter for the Amiga

PostScript is quickly becoming the industry-standard language for describing page layouts. Many dazzling effects can be achieved with PostScript. Now you can learn this increasingly important language using **PrintScript** and your Amiga!

While most desktop publishing programs can output PostScript, there are many additional features of PostScript that these programs do not utilize. With **PrintScript** you can learn how to control PostScript to create spectacular effects-whether you intend to use a 1200dpi typesetter or **PrintScript** and your own printer.

With **PrintScript** you can learn about the PostScript operators, the operand stack, dictionaries, transforms, and so on. You can develop PostScript routines that can be downloaded to PostScript printers and used to enhance your printed documents. Do your PostScript development on your Amiga, not at $$$ per page at a service bureau!

Pixelations

Pixelations, Inc.
P.O. Box 547
Northboro, MA 01532
508-393-7866

PostScript is a trademark of Adobe Systems, Inc.

3D OPTIONS

CONVERT IFF BITMAPPED PICTURES TO 2D AND 3D OBJECTS!!!

Now You Can Make Use of All Those DPaint and Digitized Pictures in Your Favorite 3D or CAD Package

VIDEOSCAPE 3D, AEGIS DRAW, INTROCAD, MCAD, PROFESSIONAL PAGE, POSTSCRIPT DXF (AUTOCAD...) & More.

TO ORDER CALL
1 800-628-2828 Ext 829

We accept VISA, MASTER CARD and AMERICAN EXPRESS

Only $49.95

Or Send Check or Money Order to:
Rainbows Edge Productions
4412 4th Avenue Suite 2
Brooklyn, NY 11220

All product names are Trademarks of their respective companies

The BEST AMIGA DISKS

Quality User Supported Software works with all Amiga systems

$7 Each Buy 1-4 Disks

$6 Each Buy 5-14 Disks

$5 Each Buy 15 or more Disks

So easy to Use

- ▼ No Computer Experience Necessary
- ▼ Instructions for each Order
- ▼ FREE Phone Support

So Easy to Order

- ▼ FREE Membership ▼ FREE 800-# for Orders
- ▼ FREE Same-day Shipping
- ▼ UPS 2nd Day Air Service when you need it Now!
- ▼ FREE Catalog ▼ Your Satisfaction Guaranteed

THE TOP 10

☐ **#131 PacMan '87** - Great sound and graphics. Adds new elements to PacMan. Saves Top 10.
☐ **#127 Wheel of Fortune** - A great computer version for multiple players. It even talks.
☐ **#23 Monopoly** - Enjoy great graphics and sound while playing three tough computer opponents.
☐ **#37 Business Programs** - Included are an address book, an amortization program, a talking mail manager, and a label printer.
☐ **#27 Amoeba Invaders** - A better Space Invaders!
☐ **#140 Virus Killer** - Everyone needs this! Makes it easy to detect and eliminate known viruses.
☐ **#115 Word Processor** - Lots of features.
☐ **#134 Applications** - Label maker/printer, grocery list maker, and AMIGazer - a star viewing program.
☐ **#142 Q-Bert** - Like the popular arcade version.
☐ **#139 Bull Run** - Great Civil War strategy board game, with impressive graphics and sound.

1-800-444-5457 for Orders

THE BEST OF THE REST

503-664-5953 - Foreign Orders

BUSINESS/HOME

☐ **#116 Business II** - VC-Spreadsheet, HP-10c calculator, and several diversions for when the boss isn't looking!
☐ **#117 Business III** - DataBase, a bunch of great new fonts, RSLClock-great clock utility, AmigaSpell.

LANGUAGES

☐ **#9 FORTH** - Two versions of the FORTH programming language.
☐ **#50 XLISP** - A version of LISP, the artificial intelligence language. Includes documentation.

COMMUNICATIONS

☐ **#4 Communications** - Starterm and Aterm are both included. Both run from Workbench or CLI and are comparable to communications packages selling for $50+. Full control of baud rates, phone directory, all protocols, auto chop of files, and many other extras. Works well with the Avatex 1200 modem and others. Source code included.
☐ **#90 Modem Madness!** - Terminals include StarTerm, ATerm, and Kermit. Telecommunication utilities such as Archive are also here. Other types of programs and utilities are also included in the price of admission.

UTILITIES/APPLICATIONS

☐ **#60 AmigaBasic** - Two programs that are truly of commercial quality, Cell-Animate and Graphit. Some Deluxe Paint picture files are also included.
☐ **#97 Tutorial Disk1** - A disk full of information and programs to instruct Amiga Programmers and users. Several C & ASM source files are included.
☐ **#98 Tutorial Disk2** - More of the best of Amiga Information.
☐ **#101 Utilities** - Many new utilities like Timeset - a time setting utility and DirCopy-a great copier (very quick), and about a dozen more. Some new fonts are also included on this disk.
☐ **#105 Potpourri I** - This disk contains several different kinds of programs, some of the highlights are: PopCLI2-evokes a new CLI window at the press of a button; PSound-sample sound recorder and editor; 3-D Breakout; DiskCat-catalogs and organizes disk files; IconMaker-makes icons for most programs so that they can run from Workbench; FKey-template maker.
☐ **#129 Amiga Utilities II** - A hard disk backup; Target-sounds a gunshot whenever the left mouse button is pressed; Dpaint Tutor; WinSize-change window size from CLI easily, and lots more.
☐ **#132 Videomaker Utilities** - This disk is packed with utilities to make your desktop videos easier to produce and more professional looking.
☐ **#133 DOS Helper** - A program designed to help you with the AmigaDOS commands. Can be activated from icon of the CLI. Supports multitasking, so that you can refer to it when you need it. As usual, there are other good programs included on the disk.
☐ **#135 Applications II** - Long Movie-plays several IFF pictures in fast succession, creating animation. QuickBase-a mail manager DBase. Persmait-a DataBase for keeping records of friends, family, associates, customers or employees. MORE.
☐ **#146 Calendar** - A very good personal calender for birthdays, holidays, meetings, bills and other events. Excellent graphics. Calendar program also has a diary. Other programs include some graphics and Checkbook.
☐ **#150 Textcraft+ Demo** - A demo of a very good word-processing program of high quality. Menu driven. Has a lot of helps to show you how. Textcraft+ also has a Speller Check available.

SOUND/MUSIC

☐ **#18 Future Sound Demo** - Another great sound demo of digitized sound. Includes the wicked witch of OZ, breaking dishes, sea gulls, car crash, ducks, others.
☐ **#30 Super Sounds** - Great digitized sounds from movies like Star Trek, 2001, James Bond movies, Star Wars, and Starman, Is it real or is it the Amiga?!!!
☐ **#77 Instruments** - Turn your keyboard into 25 different musical instruments.

SLIDE SHOWS

☐ **#1 Norman Rockwell** - 17 beautiful digitized Rockwell paintings in this self-running slideshow presentation. You've got to see these!
☐ **#55 EA Demo and Polyscope** - More great graphics for your enjoyment.
☐ **#67 DPSlide8** - Over 30 slides of all sorts on this self-running slideshow.
☐ **#94 Diga-View** - This one shows the digitizing process in stages. Several good pictures are included. Other programs also included.
☐ **#95 DigaSlide11** - Another in the great series of slideshows with great artwork. Self-running with over 25 pictures.
☐ **#108B Juggler** - Famous demo that shows the beautiful graphics of the Amiga and just how powerful this program is. It is easier to run than 108A, but only has the one demo on it.

GAMES

☐ **#38 Basic GrabBag2** - Around 25 programs of various types. Many of these are must-haves. At less than $.20 each, you can't go wrong!
☐ **#44 Games3** - More great games including Life, Vegas Slot Machine, Reversi, others.
☐ **#52 Basic Games** - Tons of ABasic games - discover some treasures!
☐ **#61 ABasic GrabBag1** - Only about 100 of all types!!!!!
☐ **#102 Sinking Island** - Return to Sinking Island is an excellent adventure game. Well worth the price - hours of enjoyment!!!
☐ **#114 Potpourri X** - Othello, A key-shortcut program for AmigaDos. Various new tools, automatic printer-driver generator, much more.
☐ **#118 Great Graphic Games** - Includes Missle Command, 2-D Triclops, Cosmo-asteroids clone, BrakeOut, Yatzee, Hack and more.
☐ **#121 Backgammon** - A great game from David Addison.
☐ **#122 Solitaire** - Two versions by David Addison.
☐ **#123 Cribbage** - Take on the computer or a friend.
☐ **#124 Milestone** - A great computer version of Miles Bournes by the author of Monopoly for the Amiga, David Addison.
☐ **#125 Othello** - A great 3-D version of this popular game.
☐ **#128 Space Games** - Cosmoroids (like asteroids) and Gravity Wars highlight this disk just full of games.
☐ **#137 Blackjack** - A full-featured game which allows pair-splitting, double-down, etc. Bandit-play the slot machines without going to Vegas!!! More.
☐ **#141 Dominoes** - Dominoes game with great graphics. Also Tic Tac Toe, Drawing and Molecules programs.
☐ **#147 Jackland/Graphics** - Adventure clue game. Also some great pictures (graphics), a useful utility called Quickbase, and a fun program called Thingies which you will enjoy!!!
☐ **#148 Boulder Dash** - Very popular game with excellent graphics and has several challenging levels. This disk is full- It has Othello, Life3 and many useful utilities.
☐ **#151 4 in a Row** - A fun, but challenging game you play against the computer. There is an excellent Demo "MandFXP-D3", a utility or 2 and the fun TARGET - A weirdo thingy.

MISCELLANEOUS

☐ **#88 Amiga Basic Programs** - Over 50. Games, utilities, applications, entertainment, and finance. Also included is a program that allows you to use IFF files in your Amiga Basic programs.
☐ **#119 mCAD** - A full-featured computer-aided design program.
☐ **#136 Graphics2** - Border Set-useful for desktop publishing and video, making cards, coupons or menus, and your own artwork. Xicon-allows you to run AmigaDOS commands or programs from Icon.
☐ **#144 Christmas Animations** - 10 beautiful scenes and graphics with sound. Great scenes include: Lit Candle, Elves, The Christmas Tree, The Manger, Season's Greetings, Holly Wreath, Chimney Smoke, Church Bells, and Walking Home.
☐ **#145 Animations 3** - More Great Animations. 3 very good demos plus 3 workbench pictures and Blobs.

Name _____
Address _____
City _____ State ____ Zip ____
Phone (____) _____
Visa/MC ____-____-____-____
Sig. _____
Exp. Date _____ ☐ Send Catalog

Disks Ordered ____ × ____ $____
☐ SHIPPING Free U.S. $____
Canada - 25¢ per disk Foreign - 50¢ per disk
☐ COD $____
(add $4 if you require COD)
☐ U.P.S. 2nd Day Air (Add $3) $____
TOTAL $____
☐ Check/MO ☐ Visa/MC ☐ COD

Software Excitement!

P.O. Box 5069 • Central Point, OR 97502

VISA ORDER TODAY MasterCard

1-800-444-5457

Reader Service Card

Fill out this card (please print) or attach your peel-off mailing label, and circle the reader service number for the ads that you'd like more information about (see Advertiser Index below).

Name _____

Company _____

Address _____

_____

Postal/Zip Code _____

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

TA V2I2

Advertiser	Page	Reader Service #
Antic Software	33	1
ASDG Inc.	11	2
Comp-U-Save	68	3
Creative Computers	28-29	4
CSA (Computer Systems Associates)	5	5
Discovery Software	Outside Back Cover	6
Dixalations	79	7
Electronics 2001	75	8
Express-Way Software Inc.	11	9
Fisher's Computers & Software	75	10
Gramma Software	27	11
Great Valley Products	Inside Back Cover	12
Inkwell Systems	7	13
Inovatronics Inc.	1	14
Intercomputing Inc.	17	15
Interface Technologies	75	16
Lake Forest Logic Inc.	67	17
LogiComp Software	71	18
MCP Associates Inc.	67	19
Metadigm	27	20
Microcube Corporation	51	21
Micromaster Inc.	16	22
Microsmiths Inc.	6	23
Newtek Inc.	32	24
OTG Software	5	25
PC Plus	16	26
Practical Solutions	27	27
Rainbows Edge Productions	79	28
Rick Stiles (UEdit)	13	29
Safe Harbor Computer Supplies	51	30
Software Excitement	80	31
VidTech International Inc.	Inside Front Cover	32
William S. Hawes	71	33

for the AMIGA Transactor

The Magazine for Amiga Programmers

Please print your name and address below (or transfer your mailing label)

Check here for change of address ☐

Name: _____

Address: _____

City: _____ *Province/State:* _____

Zip/Postal: _____ *Phone:* _____ *CompuServe ID:* _____

NOTE: Prepayment required. Purchase orders will be accepted ONLY if accompanied by payment.

Subscribers only: please enter subscription number from mailing label ☐

Subscription Order Section

PLEASE NOTE THAT TRANSACTOR FOR THE AMIGA IS NOW PUBLISHED MONTHLY

	Canada ($Cdn.)			U.S.A. ($US)			Foreign ($US)			Air Mail ($US, overseas only)		
	Mag	Disk*	Both*	Mag	Disk	Both	Mag	Disk	Both	Mag	Disk	Both
12 issues	☐35.00	☐123.00	☐158.00	☐27.50	☐101.00	☐128.50	☐38.50	☐119.50	☐157.50	☐ 73.00	☐ 145.00	☐ 218.00
24 issues	☐65.00	☐220.00	☐270.00	☐49.00	☐160.00	☐190.00	☐69.00	☐220.00	☐290.00	☐ 135.00	☐ 250.00	☐ 390.00

* Ontario residents please add 8% of disk subscription price for provincial sales tax. No PST on magazine subscriptions.

☐ **Please send Microfiche instead of magazines** ☐ **Send me info on Transactor Classic (C64/128)**

Combo Subscription Bonus!

If you are subscribing to BOTH the magazine and the disk, check one of the following FREE gifts:

☐ **Transactor Classic Amiga Disk #1** ☐ **Fish Catalog Disk** ☐ **Extended Subscription (13 Issues)**

Product Order Section

Ontario residents please add 8% PST on all items

$Cdn.	$U.S.	
☐ 14.95	12.95	Transactor Classic Amiga Disk #1
☐ 14.95	12.95	Transactor Classic Amiga Disk #2

Fish Disks $6.95 US, $7.95 Cdn. each

Note: Disk numbers not yet released may be ordered in advance - simply add numbers to end of list. Disks 57, 80 and 88 are not available.

☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7	☐ 8	☐ 9	☐ 10	☐ 11	☐ 12	☐ 13	☐ 14
☐ 15	☐ 16	☐ 17	☐ 18	☐ 19	☐ 20	☐ 21	☐ 22	☐ 23	☐ 24	☐ 25	☐ 26	☐ 27	☐ 28
☐ 29	☐ 30	☐ 31	☐ 32	☐ 33	☐ 34	☐ 35	☐ 36	☐ 37	☐ 38	☐ 39	☐ 40	☐ 41	☐ 42
☐ 43	☐ 44	☐ 45	☐ 46	☐ 47	☐ 48	☐ 49	☐ 50	☐ 51	☐ 52	☐ 53	☐ 54	☐ 55	☐ 56
☐ 58	☐ 59	☐ 60	☐ 61	☐ 62	☐ 63	☐ 64	☐ 65	☐ 66	☐ 67	☐ 68	☐ 69	☐ 70	☐ 71
☐ 72	☐ 73	☐ 74	☐ 75	☐ 76	☐ 77	☐ 78	☐ 79	☐ 81	☐ 82	☐ 83	☐ 84	☐ 85	☐ 86
☐ 87	☐ 89	☐ 90	☐ 91	☐ 92	☐ 93	☐ 94	☐ 95	☐ 96	☐ 97	☐ 98	☐ 99	☐ 100	☐ 101
☐ 102	☐ 103	☐ 104	☐ 105	☐ 106	☐ 107	☐ 108	☐ 109	☐ 110	☐ 111	☐ 112	☐ 113	☐ 114	☐ 115
☐ 116	☐ 117	☐ 118	☐ 119	☐ 120	☐ 121	☐ 122	☐ 123	☐ 124	☐ 125	☐ 126	☐ 127	☐ 128	☐ 129
☐ 130	☐ 131	☐ 133	☐ 134	☐ 135	☐ 136	☐ 137	☐ 138	☐ 139	☐ 140	☐ 141	☐ 142	☐ 143	☐ 144
☐ 145	☐ 146	☐ 147	☐ 148	☐ 149	☐ 150	☐ 151	☐ 152	☐ 153	☐ 154	☐ 155	☐ 156	☐ 157	☐ 158
☐ 159	☐ 160	☐ 161	☐ 162	☐ 163	☐ 164	☐ 165	☐ 166	☐ 167	☐ 168	☐ 169	☐ 170	☐ 171	☐ 172

The Transactor for the *AMIGA* Disk $12.95 US, $14.95 Cdn. each

(these disks correspond to issues of **Transactor for the *AMIGA***)

☐ Vol. 1, Iss. 1 ☐ Vol. 1, Iss. 2 ☐ Vol. 1, Iss. 3 ☐ Vol. 1, Iss. 4 ☐ Vol. 1, Iss. 5 ☐ Vol. 1, Iss. 6

☐ Vol. 2, Iss. 1 ☐ Vol. 2, Iss. 2

Add flat $2.00 shipping & handling for any number of items ordered from above box.

Back Issues, Microfiche: $4.50 U.S., $4.50 Cdn. (MF=Microfiche, filled box=SOLD OUT)

☐V1#1 ☐MF ■V1#2 ☐MF ☐V1#3 ☐MF ☐V1#4 ☐MF ☐V1#5 ☐MF ☐V1#6 ☐MF

☐V2#1 ☐MF ☐V2#2 ☐MF

Ontario residents please add 8% provincial sales tax for any items ordered from this box.

☐ Visa ☐ MasterCard ☐ Cheque/MO enclosed Cheque #_____ Dated ____/____/____ Amount _____

Acct. # _____ Expires ____/____ Signature _____

Please allow 6-8 weeks for delivery in the U.S., 4-6 weeks in Canada.

FOLD ALONG SCORES WITH CORRECT REPLY CARD ON OUTSIDE. USE TAPE, PLEASE - NO STAPLES OR G

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 390 BUFFALO, NY
POSTAGE WILL BE PAID BY ADDRESSEE

The Transactor
P.O. Box 338 Station 'C'
Buffalo, NY, 14209-9990

place stamp here

Transactor
10-85 West Wilmot Street
Richmond Hill, Ontario, Canada
L4B 1K7

Use our convenient...

TOLL-FREE ORDER LINE
1-800-248-2719 Extension 911

Here's a convenient way to order magazine or disk subscriptions and other products. Nothing to fill out or drop in the mail! Just call the toll-free number above and have your credit card number ready. Available in the United States only.

NOTE: The toll-free number can be used for ordering only; for subscription queries, etc., call or write to our office directly.

Business Reply Mail
No Postage Stamp Necessary
If mailed in Canada

Postage will be paid by:

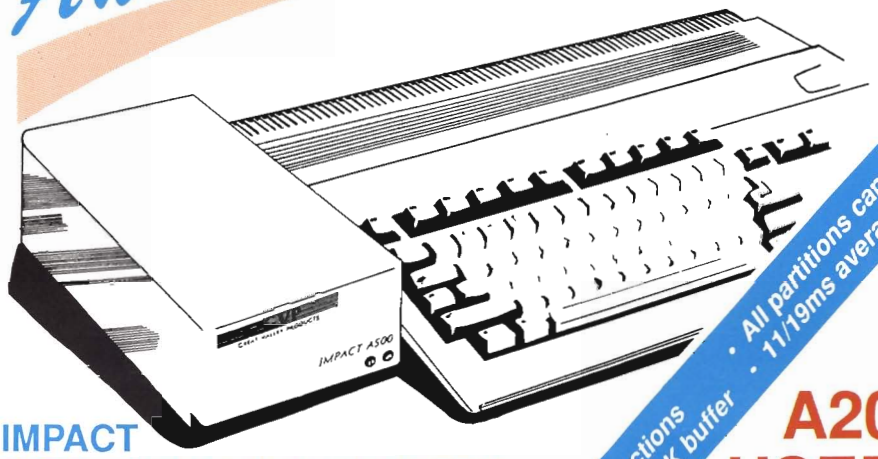
Canada Post Postes Canada 46487

The Transactor
501 Alden Road
P.O. Box 3250
Markham Industrial Park
Markham, Ontario
L3R 9Z9

AMIGA 500 USERS!
NOW GET A2000 POWER!!

Announcing

THE WORLD'S FIRST AUTOBOOTING HARD DISK FOR THE AMIGA 500



• All partitions can be FFS with V1.3 AUTOBOOT
• 11/19ms average access time on 40Q & 80Q models

IMPACT A500 – HD/RAM SUBSYSTEM

offers everything any A500 owner has ever dreamed of – in one compact, easy-to-install, add-on subsystem; provides the same high performance as our top-of-the-line A2000 controller

IMPACT™ A500 – HD/20
A500 – HD/30
A500 – HD/40Q
A500 – HD/45
A500 – HD/80Q

Available with a built-in 20, 30, 40, 45 or 80 Mega Byte Hard Drive and an optional 2MB plug-in FAST RAM/AUTOBOOT expansion module.

• Internal & external SCSI connections
• DMA data transfer to onboard 16K buffer

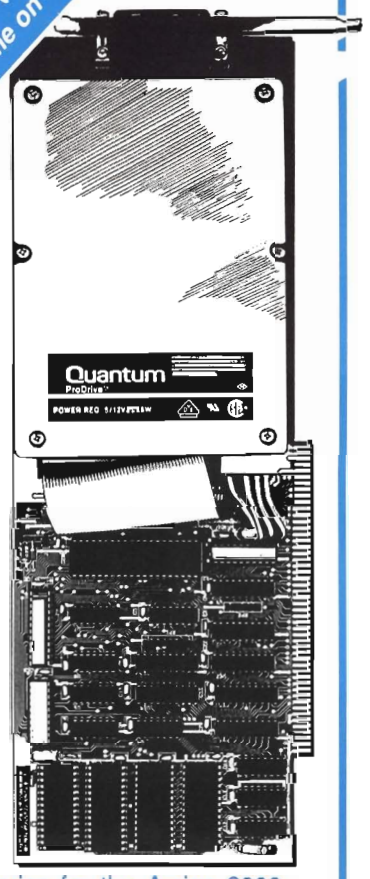
• AUTOBOOT directly from hard disk
• Connect up to 6 SCSI peripherals

A2000 USERS!

Still

THE MOST POWERFUL HARD DISK SOLUTION FOR YOUR A2000!

IMPACT™ A2000 - HC/30
A2000 - HC/40Q
A2000 - HC/45
A2000 - HC/80Q
A2000 - HC/100



The GVP IMPACT-HC "Harddisk-on-a-Card" series for the Amiga 2000 matched with the 3.5" Quantum ProDrive hard disk drives, provide turbo performance and unbeatable value at list prices as low as $18.00 per Maga Byte! The Quantum ProDrives provide average access times as low as 11ms with their built-in 64KB read-ahead cache and offer a MTBF of 50,000 hours which is almost twice the industry average.

Now Shipping **AUTOBOOT EPROMS**

GVP

GREAT VALLEY PRODUCTS INC.

225 PLANK AVENUE, PAOLI, PA 19301 TEL (215) 889-9411 FAX (215) 889-9416

SWORD OF SODAN™

Grasp the Sword of Sodan and feel the true power of arcade action!

Creaking, groaning, shrieking, teeth clenching arcade realism.
Deep dark forests.
Misty, eerie graveyards.
Treacherous, guard-filled streets.
Dare to enter the evil castle, Craggamoore.
Wield your sword through its winding corridors and rank rotting dungeons.
Your story will be the greatest ever told — Your reward will be the greatest ever known!

FEATURES:

- ***Battle with massive animated characters that stand 2/3rds the size of the screen.***
- ***Fight as Hero or Heroine through 11 incredibly challenging levels!***
- ***Study your past performance in Replay Mode! (extra memory required)***

Packed into 3 disks with more than 4 MEGABYTES of detailed graphics and digitized sounds!

Available for Amiga™, Coming soon for IBM® and compatibles, Apple II GS.

AMIGA screens shown.

DISCOVERY SOFTWARE

Fight your way to your local software retailers. Or, call 1-800-342-6442 for VISA and MasterCard Orders. For check or money orders, send $49.95 plus $3 shipping and handling to DISCOVERY SOFTWARE INTERNATIONAL INC. 163 Conduit Street, Annapolis, MD 21401. Allow 4-6 weeks for delivery. Sword of Sodan is a trademark of Discovery Software International, Inc. Amiga is a trademark of Commodore Amiga, Inc. IBM is a registered trademark of International Business Machines, Inc.

Dorian 1988 ©